INTERACTIVE MULTIMEDIA MAIL SCENARIOS:

AN IMPLEMENTATION

By

JEFFREY B. HOLLAND

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1994

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1997

INTERACTIVE MULTIMEDIA MAIL SCENARIOS:

AN IMPLEMENTATION

Thesis Approved:

H. Lu

Thesis Advisor

J Chandler

Blayne E. Mayfield

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to sincerely thank my thesis advisor, Dr. Huizhu Lu, for her outstanding help, guidance, and support. I would also like to thank Dr. Lu for her wonderful inspiration, understanding, and kindness. Without Dr. Lu's willingness to go beyond the ordinary call of duty, it would not have been possible for me to complete this thesis. I would also like to sincerely thank my other committee members, Dr. Blaine Mayfield and Dr. John Chandler, whose guidance, help, encouragement, and understanding are also invaluable.

I would also like to thank all of my family and friends who have stood by me as I devoted almost all of my time to working and completing this degree over the past few years. I would especially like to thank my mother and my sister who have always stood by me even in the most difficult times, providing unprecedented love, understanding, and support.

Finally, I would also like to that the rest of the faculty and staff of the Computer Science Department who have helped me so much during these last few years of study.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Multimedia collaboration is a form of collaboration that uses computer networks and multimedia to facilitate cooperative work between geographically widespread parties. Multimedia collaboration systems include audio conferencing, video conferencing, chalkboarding, and multimedia mail. Chalkboarding, also referred to as whiteboarding, is a form of collaboration where the collaborating parties use shared images (chalkboards) to facilitate a discussion over a computer network. Multimedia mail can provide an alternative means for collaboration mechanisms such as chalkboarding when there is difficulty staging an interactive real time communication between parties. In order for multimedia mail to provide an effective alternative, sufficient demonstrative power must be built into mail messages.

Multimedia mail allows communication of messages that consist not only of text, but also other media such as images, audio, and video. Even though most electronic mail messages are static after creation, computational power may be built into electronic mail messages. Computational mail is electronic mail where messages contain programs to be executed during the delivery or viewing of messages. Computational mail is also known as enabled mail.

A multimedia mail scenario service is a specialized computational mail service that is used to exchange multimedia scenarios between communicating parties. In multimedia mail scenario services the multimedia parts contained in a message are displayed for the message receiver in the manner indicated by the accompanying scenario

code. In previously proposed scenario services scenario code was specified in a scenario language used to specify the timing and ordering for the playback of the multimedia message body parts.

Kervella *et al.* [KGH94] and Agastani [Aga95] have both proposed multimedia mail scenario services based on the specialized scenario languages that they proposed. Kervella *et al.* [KGH94] proposed a scenario service based on the X.400 mail standard. The scenario service that Agastani proposed and implemented was based on the Multipurpose Internet Mail Extensions (MIME) standard.

We propose a new scheme for a scenario messaging service based on the MIME standard [BF96a]. The new scheme has two main goals. The first goal is to provide a scenario service that is a more effective multimedia collaboration tool. The second goal is to provide a more portable and flexible scenario service.

Timing for multimedia scenarios in existing messaging services could be specified only for predetermined intervals when a scenario is created. These types of wait times cause a pause for a predetermined specific time interval. If a part of a multimedia scenario contains complex images or other material that take varying amounts of time to view, this type of timing mechanism will not be effective. To overcome this problem, the service that we propose adds user interaction capability to the scenario service to allow pre-specified portions of the scenario to be viewed for times determined by viewer interaction. This is achieved through the addition of slide show capability to the scenario service model. Slide show capability allows a scenario viewer to move between the parts (slides) of a message at the viewer's discretion. To improve effectiveness and ease of use, the service allows the user to move not only to the next

part of a message, but also to the previous part.

A second main addition to the scenario service scheme that we propose is reference pointer use. The proposed scenario service provides the capability for the composer of a message to specify reference pointers to be displayed on a media, such as an image, to emphasize parts of the media. A pointer can be used to facilitate a discussion by visually emphasizing the referenced part of the media.

To provide a messaging service that is both portable and flexible, we implement the service based on the Java language and "plug in" components. Use of the Java language provides the portability, multimedia support, and applet security safeguards that are built in to the Java language. To take advantage of this, scenarios composed in the service are implemented as Java applets. This avoids the need for a specialized interpreter, as well as facilitates integration of the scenarios with the World Wide Web since the scenarios may be presented in a Java enabled Web browser.

To take advantage of the portability and multimedia support that Java provides, we have also implemented the scenario service application in the Java language. We implemented the proposed service using a GUI based program that facilitates easy composition, sending, receiving, and presentation of multimedia scenarios. In our implementation, we include support for image, audio, and text media.

The remainder of this thesis is organized into the following chapters:

- Chapter 2: Contains an overview of previous work related to multimedia mail systems.
- Chapter 3: Contains the proposed scenario service scheme.

- Chapter 4: Contains the implementation of the proposed scheme.
- Chapter 5: Contains an overview of the results of implementing the proposed scheme.
- Chapter 6: Contains a summary of the thesis and suggestions for future work.

CHAPTER 2

LITERATURE REVIEW

2.1. Message Handling System

The main goal of an electronic mail system is to convey messages from the sending party to the receiving party. Traditional e-mail messages consisted only of a header and a text body. The header of the message contains information specifying the purpose and receiver(s) of the message. The body of a message contains the information that the sending user wishes to convey to the receiving user(s).

An electronic mail system consists of User Agents (UAs), and a Message Transport System (MTS). A User Agent is software that comprises the part of the mail system that interfaces with the users. A User Agent is a program that is used to create, send, save, retrieve, and display mail messages. The MTS is used to transport messages from the sender to the receiver(s). The MTS is made up of Message Transfer Agents (MTAs). An MTA routes and relays electronic mail messages.

2.2. Multimedia Mail Standards

In order to be able to send e-mail containing more than a text based body, multimedia mail standards began to be developed. There are two widely used standards in multimedia mail. One of them is X.400 [CG95] developed by CCITT. The other widely used standard is Multipurpose Internet Mail Extensions (MIME) [BF96a]. MIME is a

standard for Internet mail defined by the Internet Engineering Task Force (IETF) that allows a wide variety of media to be included in an e-mail message, such as images, audio, video, text, and applications. MIME is an extension of ordinary Internet mail as defined in RFC 822 [Cro82]. The MIME standard is in very widespread use, being used for more than just Internet mail applications.

MIME defines seven valid Content types used to specify the types of data that can be included as mail body parts. The seven content type values defined by MIME are:

1. ***text***. The text type is the default content type. Text formats other than regular U.S. ASCII may be used by specifying the appropriate parameters [BFr96].

2. ***image***. The image type is used for still images. Subtypes are image format names. The two image formats that are defined by the core MIME standard mail readers are GIF and JPEG. These two subtypes are specified by "image/gif" and "image/jpeg" respectively.

3. ***audio***. The audio type is used for audio information. Sub-types are audio format names. MIME defines the initial subtype "basic" for single channel 8000 HZ u-law audio data. It is represented by the sub-type "audio/basic"

4. ***video***. The video type is used for video information. MIME defines the sub-types "video/mpeg" and "video/h261" for MPEG and H.261 videos, respectively.

5. ***message***. The message type is used to encapsulate an entire RFC 822 message for inclusion.

6. ***multipart***. The multipart type is used to include several body parts that may be of different types and sub-types into a single RFC 822 message body. Boundary

parameters are used to separate the different body parts within the multipart part. These parameters are specified in the multipart content-type field.

7. ***application***. The application type is used to include body parts that do not fall into the other six content-type categories. This type may be used to enclose mail application language code that will be executed during message delivery or viewing. MIME defines the "application/postscript" subtype for the transport of postscript documents.

MIME is an evolving standard and additional content types may be proposed to IETF. Implementors are generally required to register new subtypes with the Internet Assigned Numbers Authority to avoid naming conflicts. However, private subtypes may be used without registration. Private subtypes begin with "x-".

2.3. Computational Mail

Most e-mail systems used today are static and do allow computational power to be enclosed in a message. Even though most multimedia mail software is not active, e-mail messages may contain programs to be executed as messages are delivered or viewed. Electronic mail that includes computational power in messages to be viewed is known as computational mail or enabled mail. Rose and Borenstein [RB94] had proposed a general model for enabled mail to give computational power to an electronic mail system. Enabled mail adds computational power at three phases in the message delivery [RB94]:

1) **delivery time** - Computation is performed right before the message crosses the delivery slot.

2) **receipt time** - Computation is performed right after the message crosses the delivery slot.

3) **activation time** - Computation is performed when the recipient views a received message. Mail containing this type of computational power is known as *active mail*.

Borenstein [Bor94] developed the Safe-Tcl language to be used in enabled mail. This language was based on the Tcl language and contains no constructs that can be used to do harm with e-mail messages.

Danvind and Mattson [DM95] had developed a general purpose computational mail tool based on Java. This tool allowed a user to incorporate a manually written Java program into a MIME mail message. The messaging system that we propose uses a different approach which does not require a user to program the Java applets that form the basis of our messaging system.

2.4. Java

Java is an interpreted but high performance object oriented language. Java has strong networking capabilities and has syntax that is similar to that of C++. Java has many advantages. Java supports graphical user interfaces, multimedia, and multithreading. Another major advantage is that Java is a highly portable language. This portability is achieved by compiling Java source code into a virtual machine code called Java

bytecodes. The bytecode instructions have nothing to do with any particular architecture and can be run on any machine with a Java interpreter.

Another advantage is that Java may be used to produce both applets and applications. Applets are Java programs that run on the html World Wide Web pages that they are associated with (embedded inside). The Web page that an applet is embedded in contains html statements that indicate:

1. the name of the compiled applet file (the class file).
2. the location of the compiled applet file.
3. how the applet sits on the Web page.

Java applets can be viewed by invoking an applet viewer on the html page that embeds the applet. Java was designed for network use with security in mind. Java Applets contain no mechanisms that can be used to do harm to the client machines that they run on.

2.5. Scenario Services

Traditional multimedia mail was limited since it did not provide any mechanism to specify timing and ordering in the playback of multimedia messages. Kervella *et al.* [KGH94] proposed a multimedia mail scenario service based on the X.400 mail standard. A scenario service is an electronic mail service where the sender of a message specifies a multimedia scenario to be viewed by the receiver of a message. Multimedia mail scenario services are a specialized form of computational mail where specialized

languages (scenario languages) have been used to specify timing and ordering for the playback of multimedia mail messages. The scenario language that Kervella *et al.* [KGH94] proposed consisted of substitution statements and synchronization statements. The substitution statements for this language are used to specify alternative body parts that may be displayed if the viewer of a message does not have the capability (hardware or software) to display the intended body part. The synchronization statements are used to specify timing and ordering for displaying the message body parts for the receiver. The language proposed by Kervella *et al.* is shown in Figure 2.1 and Figure 2.2.

Substitution Statement	Action
IFNOT<bodypart name1> THEN <bodypart name2>	presents <bodypart name2> if the user agent software cannot present <bodypart name1>
IFNOT<bodypart name1> THEN SUBSTITUTE	presents a substitute text if the User Agent software cannot present <bodypart name1>
<statement>&<statement>	combines statements that are atomic

Figure 2.1. Substitution Statements [KGH94]

Synchronization Statement	Action
<bodypart name 1>	presents <bodypart name1>
WAIT_END <statement>	waits up to the end of <statement>
WAIT(<value>, <statement>)	waits the number of seconds indicated in <value> before presenting <statement>
SAME_START(<statement>, <statement>)	indicates that the <statements> have to begin at the same time
SAME_END(<statement>, <statement>)	indicates that the <statements> have to finish at the same time
SAME_LIMITS(<statement>, <statement>)	indicates that the <statements> have to start and finish at the same time

Figure 2.2. Synchronization Statements [KGH94]

This proposed service was never fully implemented, but Agastani [Aga95] had proposed a similar scenario service that he implemented based on the MIME mail standard. Agastani [Aga95] proposed the language shown in Figure 2.3 and Figure 2.4 to specify the timing and ordering for the playback of MIME multimedia mail body parts:

Substitution Statement	Action
IFNOT<bodypart1[/geom]> THEN <bodypart2[/geom]>	presents <bodypart2> if the user agent software cannot present <bodypart1>
IFNOT<bodypart1[/geom]> THEN SUBSTITUTE	presents a substitute text if the user agent software cannot present <bodypart1>

Figure 2.3. Substitution Statements [Aga95]

Synchronization Statement	Action
<bodypart[/geom]>	presents <bodypart>
WAIT(<value>, <bodypart[/geom]>)	waits the number of seconds indicated in <value> before presenting <bodypart[/geom]>
PAR{<bodypart[/geom]>|WAIT(<value>, <body part[/geom]>)>;...}	indicates that <body parts[/geoms]> are presented in parallel maybe with the waiting time for certain body part name.

Figure 2.4. Synchronization Statements [Aga95]

The [/geom] parameter used in these statements is used to resize and / or position the window in which the media is displayed.

This service was implemented using the Tcl/Tk scripting language to implement the user interface. The user interface consisted of an X-Windows based interface that the message composer used to create the scenario with. This creation process generated the scenario language code to be interpreted when a scenario was presented for the receiver. The interpreter was implemented using the C language.

CHAPTER 3

SCENARIO SERVICE

In this thesis, we propose a new scheme for implementation of the scenario service for Internet multimedia mail based on the Multipurpose Internet Mail Extensions (MIME) standard. The proposed active mail service allows the composition, sending, and presentation of the multimedia scenarios that form the basis of the messaging system. The service is designed to be a more effective tool for multimedia collaboration by providing more effective scenarios and a portable scenario service. Increased effectiveness is achieved by the addition of features such as slide show capability and reference pointer use to the multimedia mail scenario service.

In order to increase portability and flexibility, the proposed scheme is based on the Java language and plug in components. The proposed service uses Java applet code to implement scenarios so that received scenarios may be presented in the absence of the user agent software for this service. This scenario code representation also facilitates integration with the World Wide Web browser.

3.1. Slide Show Capability

In order for the synchronization mechanisms of a scenario service to be effective, the service must facilitate both sequential and parallel playback of scenario parts. The service must also allow the composer of a message to specify a wait time to introduce a pause in the scenario. Wait times are implemented in the scenario service that Agastani

proposed through the WAIT statement in the proposed scenario language [Aga95]. Wait times are implemented in the service that Kervella *et al.* [KGH94] proposed through the WAIT and WAIT_END statements in the proposed language. These synchronization mechanisms are fundamental to the scenario model.

Pre-specified wait times, however, can not always be effective since the composer of a scenario can not always determine the time needed for a particular viewer to view a part of a scenario. Since it takes varying amounts of time to view message parts such as complex images, we propose adding one additional timing mechanism to the scenario service that allows user interaction to specify wait times as the scenario is presented. The mechanism is implemented through slide show capability which allows the viewer of a scenario to move to the next or previous part of a scenario at their own discretion. Slide show capability uses multimedia slides that allow viewer interaction to trigger movement between parts of a scenario. An example of a multimedia slide that is displayed in a Netscape Web browser is shown in Figure 3.1 below.

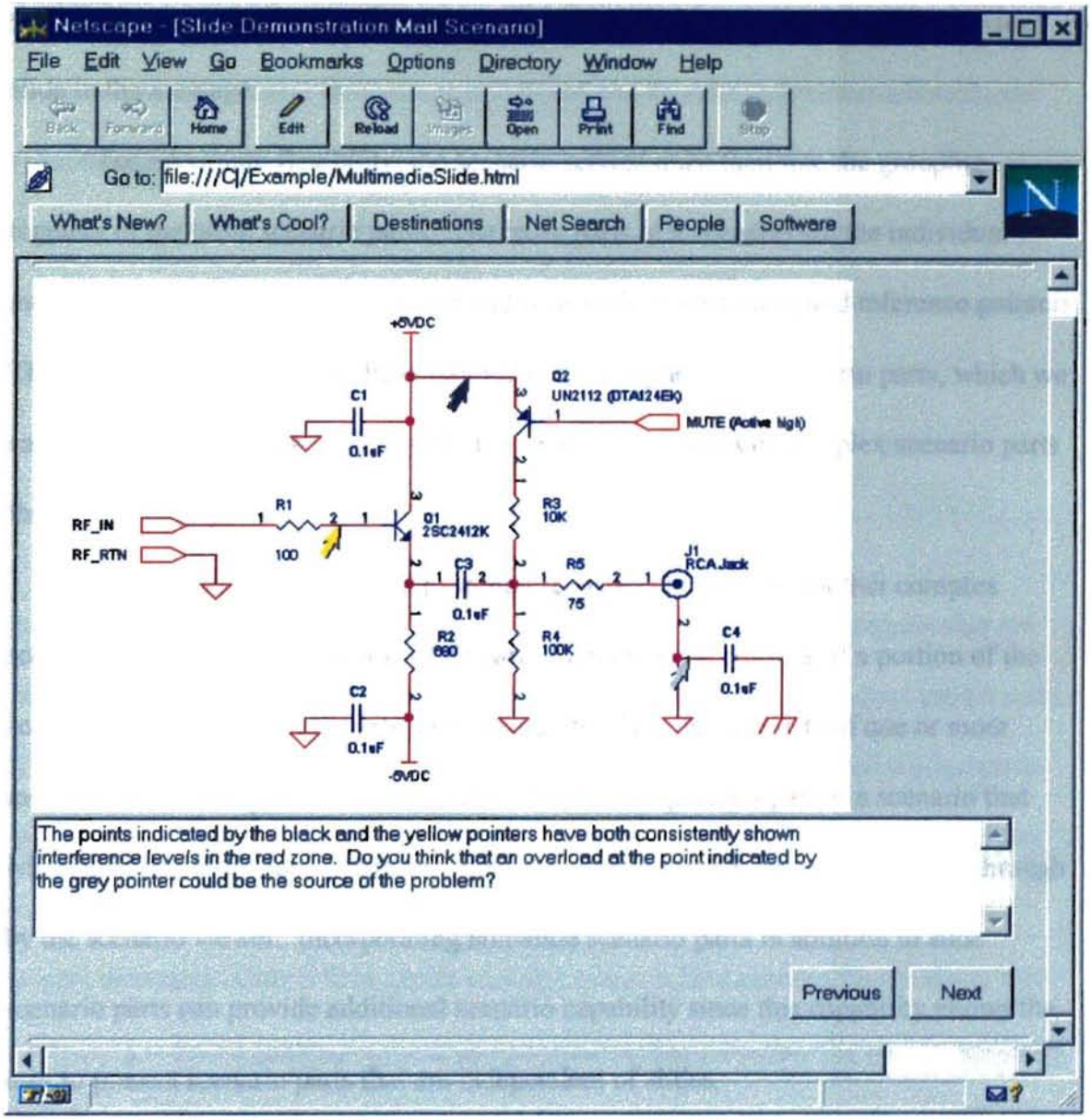

Figure 3.1. Multimedia Slide

A multimedia slide used in a scenario service consists of the portion of a scenario that is played out on that particular slide. This consists of one or more media parts that are positioned, ordered, and timed to specify a part of a scenario that will belong to that slide. The Next button displayed with a slide is used to provide movement to the next

part of the message. The Previous button is used to provide movement to the previous slide in the message.

For maximum flexibility, the scenario service must facilitate the grouping together of the basic scenario parts. The basic parts of a scenario are the individual media parts, such as images, text, and audio, as well as wait times and reference pointers. The basic scenario parts may be combined to form higher level scenario parts, which we call *complex* scenario parts. The slide is one of the two types of complex scenario parts that we propose.

To improve flexibility, the proposed scenario service uses another complex scenario part that we call the *non-slide* part. The non-slide consists of a portion of the scenario that is played out independent of a slide. This part consists of one or more media parts that are positioned, ordered, and timed to specify a part of a scenario that will not belong to a slide. Therefore, non-slide scenario parts cannot be stepped through by the scenario viewer. Incorporating non-slide scenario parts in addition to slide scenario parts can provide additional scenario capability since this capability allows the user to present scenario parts that are independent of slides.

3.2. Reference Pointers

In order to provide a more effective multimedia collaboration tool, we also propose the inclusion of reference pointers as another addition to the scenario service. The service that we propose incorporates reference pointer use which allows color coded reference pointers to be placed at specified locations on a visual media, such as an image, to

emphasize parts of the media. These pointers are similar to mouse pointers and provide visual points of reference to facilitate collaboration. In order to be most effective, the reference pointers must be separate objects from the media that they are placed on so that the underlying media remains unchanged for reuse. Since multiple points of reference may be needed in a scenario part, we propose allowing different colored pointers to be used for pointer distinction.

3.3. Scenario Representation

In order to take advantage of the portability, security, and multimedia support that the Java language provides, we propose to implement the scenario code as Java applet code. If a specialized language is used for the scenario code, then a specialized interpreter is required to execute the scenario code to present the scenario. However, implementing scenarios as Java applets removes reliance upon specialized user agent software to present messages. Only a Java applet viewer, such as a Java enabled World Wide Web browser, is required to execute the code that presents the scenario. For user friendliness, the user agent that we propose does contain the capability to automatically invoke presentation of a scenario that is received. However, even in the absence of the scenario service user agent, the receiver of a message may still view a received scenario by invoking an applet viewer on the html page that the scenario applet is embedded inside. Since applets are portable and since applet viewers are available on many platforms, this scheme makes the scenarios portable across platforms.

Since the proposed service implements scenarios using Java code, there may not

be a one to one correspondence between a statement in one of the scenario languages proposed by Kervella *et al.* or Agastani and the Java code that implements scenario functionality in the proposed service. The proposed scenario service does, however, contain all of the synchronization functionality found in the service that Agastani implemented with the addition of the slide show synchronization, as well. However, since Java has built in image and audio support, the substitution functionality can be reduced. Substitution statements were used in the scenario languages proposed by Kervella *et al.* and Agastani to present an alternative body part if the intended body part could not be presented because of lack of support for the media presentation on the viewer's machine. Since a machine where a scenario is presented may not have sound capability, we do include the ability to supply replacement text to substitute for audio in such cases.

3.4. Security

When considering a language to be used for computational mail, the security issues involved must be addressed. Any time a program is loaded from a remote site and executed on a client (local) machine, there is a threat that the program can intentionally or unintentionally do harm to the client machine. Java applets were designed to be executed on client machines without possibility of causing harm. The scenario service code is safe as can be noted by the following points:

1) Applets cannot delete files on a client machine.

2) Applets can never run any local executable program.

3) Applets cannot communicate with any host other than the server from which they were downloaded.

4) The scenario code generated in the proposed service does not allow the writing to client files.

3.5. Integration With the Web Browser

Implementing the scenarios as applets is also advantageous because this allows tighter integration with the World Wide Web browser. Since the proposed service generates both the html page and Java applet needed to present the scenario, the applet can be executed directly in a Web browser that contains support for Java. Since it is common for Web browsers to contain electronic mail capability, the Web browser can serve as the message transporter, as well. With increasing use of the Web, integration of the scenario service with the Web browser makes the service more convenient for the user.

3.6. Plug In Components

In addition to making the messages used in the service portable, another goal of this thesis is to provide a scenario service that is itself portable. To provide a service that is portable and flexible, we implemented the scenario service based on the Java language and plug in components.

There are a number of functions involved with the composition, presentation, sending, and receiving of messages. Even though some of these functions can be implemented directly as part of the main User Agent (UA) program, implementing the

functions as separate programs or modules can improve portability and flexibility. The components that the main UA program interfaces with may be specified by the user by setting parameters in a control file. The proposed scenario service is composed of the components shown in Figure 3.2 below.

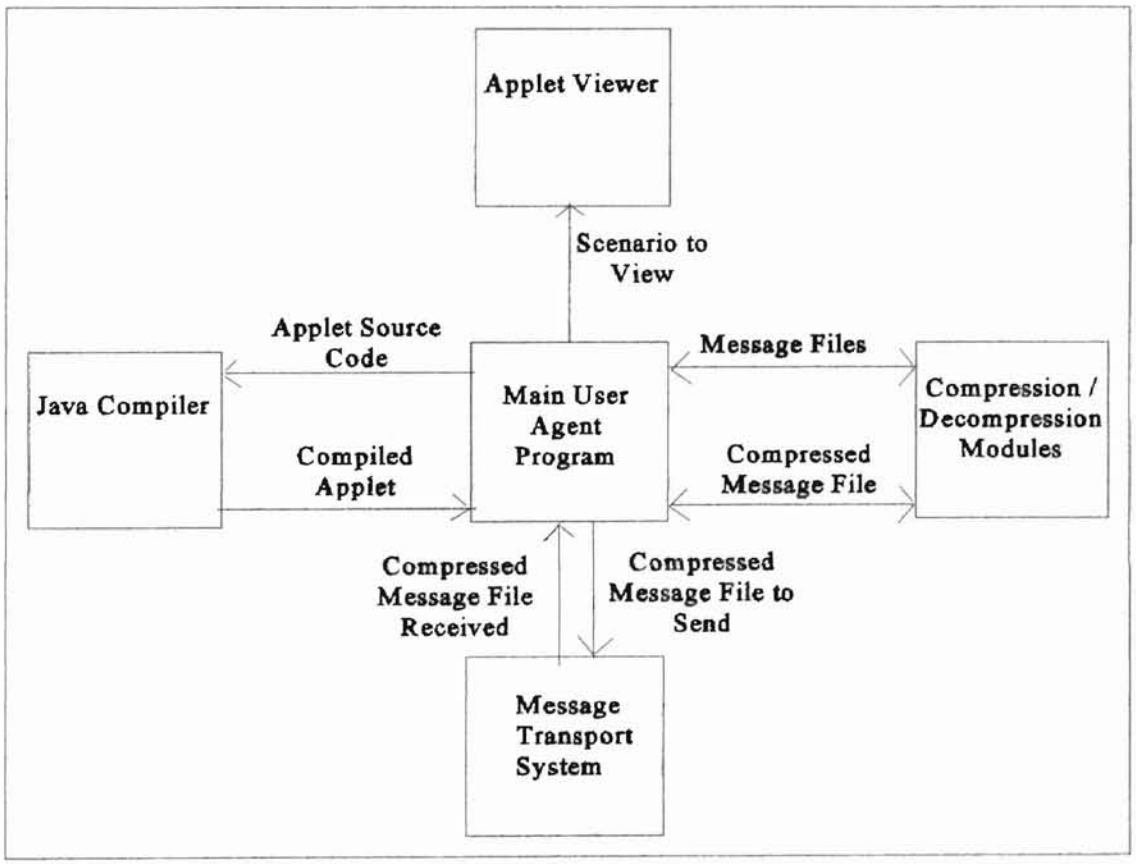

Figure 3.2. Interrelationship of Service Components

3.6.1. User Agent

The UA controls all functions related to composition and presentation of message scenarios. To ensure portability and flexibility of the scenario service, we propose to implement most of the components that the UA interfaces with as plug in components.

To implement the plug in component use, the service uses a control file where the user can specify the name of the file to execute to perform the desired function, as well as any parameters that may be needed in the command. This allows a service user to specify the program to use as a plug in component to satisfy their platform requirements and preferences.

The composition of a message requires the use of all of the scenario service components except for the Message Transport System (MTS). The User Agent must contain a graphical user interface that allows the composer of a message to easily specify a scenario. During composition of a scenario, the User Agent generates the Java applet code according to the scenario specified by the creator of the message. After the scenario has been fully specified the, the UA must generate the Java applet that will be executed to present the scenario. Since an applet must be played out in an html Web page, the program also generates an html Web page that embeds the Java applet.

3.6.2. Java Compiler

Once the Java applet and html page have been created, the applet must be compiled so that the compiled applet code can be directly executed when the scenario is presented. The Java Compiler is the component that is invoked to compile an applet generated by the User Agent.

3.6.3. Compression/Decompression Modules

In order to remove overhead involved with sending the message, we propose that the files that belong to a message be compressed into a single compressed message file before the message is sent. The compression module of the service generates a compressed

message file containing all of the files that belong to a message, such as the compiled applet, the html Web page, and all of the media files used in the scenario. This approach not only reduces the size of a message, but also simplifies sending since only a single file must be sent.

The compressed message file could be created as a self extracting archive. However, using a self extracting archive would restrict the portability of the message since the archive must contain the machine code to execute the extraction. Even though using a non-self extracting archive for the compressed message file will require the use of a decompression module, this method can increase portability. Once the message has been created, the UA relies on the MTS to transfer the message to the receiver.

3.6.4. Message Transport System

The MTS is used to transfer the MIME scenario message from the sender to the receiver. For the proposed scheme, we define the private subtype x-JavaScenario as the subtype for the messages themselves. Messages in the proposed and implemented service are assigned MIME type ***application/x-JavaScenario***.

3.6.5. Applet Viewer

The user agent relies on the applet viewer for the actual presentation of messages. When a message is received, the scenario service must invoke an applet viewer to execute the applet code to present the scenario. In order to promote integration with the World Wide Web, we propose to use a Java enabled Web browser for this task. In order to execute the scenario code, the UA must pass the html page that embeds the applet to the Web browser. The Web browser will then begin the execution of the Applet code

inside the html page.

CHAPTER 4

IMPLEMENTATION

We have implemented the proposed active mail system that allows a user to create multimedia scenario-based presentations to convey MIME messages. We implemented the scenario service on a personal computer that uses the Intel Pentium processor and the Microsoft Windows 95 operating system. However, the implementation is not platform specific. To provide a portable mail service, we implemented the scenario service based on the Java language and plug in components.

Since MIME is the most widely used multimedia mail standard, we implemented the scenario service using the MIME standard for internet mail messages. Messages in the implemented service are assigned private MIME type ***application/x-JavaScenario***.

4.1. Scenario Mail

The main program for the scenario service is called Scenario Mail. Because Java is a portable language that contains built in multimedia support, we implemented the Scenario Mail program in the Java language. The Scenario Mail program drives all scenario service activities and makes up the main portion of the User Agent. For our implementation, Scenario Mail interfaces with the Netscape Navigator 3.0 Gold Web browser for presenting scenarios, as well as sending and receiving messages.

The Scenario Mail program provides a user friendly GUI tool that the composer of a message uses to visually specify a scenario. This program also performs all of the

code generation related activities. After a scenario has been specified, the program generates the scenario code, the html page that embeds the scenario, and a file containing a list of all files belonging to the project.

4.2. Portability

To take advantage of the portability, security, and multimedia advantages provided by the Java language (discussed in sections 3.3, 3.4 and 3.5), we implemented the scenarios used in the system as Java applets. The Java applet code is generated when the composer of a message specifies the desired scenario.

The scenario service application is also designed for portability. The portability of the scenario service is achieved not only through the portability of its Java code, but also through the ability to use plug in components that interface with Scenario Mail and help it perform tasks. The control file (ScenMail.cntl) allows the user to specify parameter values such as commands for invoking the component programs. Each parameter in the control file is shown in Figure 4.1 along with a description of what each parameter specifies.

Parameter	Value Parameter Specifies
CompileCmd	Name of file executed to compile a Java applet.
CompileCmdSuffix	Value to follow the filename of the applet in the command to compile the applet.
AppletViewCmd	Name of file executed to view a Java applet.
AppletViewCmdSuffix	Value to follow the filename of the applet in the command to view the applet.
MailSendCmd	Name of file executed to invoke the mail transport program.
AppletWidth	Width of applet in pixels.
AppletHeight	Height of applet in pixels.
ApplicationWidth	Width of main window of Scenario Mail application
ApplicationHeight	Height of main window of Scenario Mail application
PointerSize	Size of reference pointers. Sizes range from 1 to 30.

Figure 4.1. Control File Parameters

By changing the corresponding control file parameters, the user can use the component programs that satisfy platform specific requirements or preferences. A parameter that has a name ending in "Suffix" is used to add any parameter or switch values that may be required at the end of the corresponding command. If the value of a parameter is not needed, then the value assigned to a parameter should be left blank. The value assigned to a parameter is specified by placing the value after the name of the parameter on the same line. At least one blank must separate the name of a parameter and its assigned value. No other characters are allowed between the name of a parameter

and its assigned value. The default values for the control file are shown in Figure 4.2. These values indicate the parameter values that we used for our Windows 95 implementation.

CompileCmd	\msdev\bin\jvc
CompileCmdSuffix	
AppletViewCmd	netscape
AppletViewCmdSuffix	
MailSendCmd	netscape mailto:
MailSendCmdSuffix	
AppletWidth	800
AppletHeight	500
ApplicationWidth	640
ApplicationHeight	480
PointerSize	15

Figure 4.2. Example Parameter Values

One additional parameter value is used by the system to indicate whether or not a machine has audio capability. Since Java does not indicate a failed attempt to play an audio clip on a machine that does not have audio capability, we use a control file value to indicate whether or not a machine has audio capability. In order to keep the applet code small and simple, this value is stored in a separate file (Audio.cntl) rather than in ScenMail.cntl. The Audio.cntl file contains a single character used to designate whether or not a machine has audio capability. If a value of "N" is successfully read as the value from this file, then this indicates that the machine does not have sound capability. In that case, the applet will present a substitute text in place of the audio if a replacement text has been specified. If a value of "N" is not successfully read, then the applet assumes

that the client machine has audio capability and will play the audio clip.

4.3. Applet Compilation

The Scenario Mail program interfaces with a Java compiler which compiles the applets that implement scenarios. The compiler converts the applet source code that is generated by Scenario Mail into Java bytecodes that can be directly executed on the receivers machine. For our implementation, we chose to use the JVC compiler that is a part of the Microsoft Visual J++ software. However, any Java compiler that is capable of compiling applets may be used by setting the CompileCmd and the CompileCmdSuffix parameters in the control file.

4.4. Compression / Decompression

Since multimedia files can be very large, the scenario service uses a compression module to compress all of the files belonging to a message into a single compressed message file to be sent. The files included in the message will consist of the following files:

1) the compiled Java applet that implements the scenario.

2) the html Web page that embeds the applet.

3) the media files for any media parts such as audio and images used in the scenario.

4) the file list file.

The file list file contains a list of the filenames of all files that are used in a message. This list is used for deletion of a message. In order to delete a message, the Scenario Mail program must know which files to remove from the local file system. The program does this by reading the file list file that is sent as part of the message. Each file named in this file is removed when a user invokes the deletion of an entire message. The user may also specify a deletion where the media files are not removed. In this case any files that are media files are not removed.

Since Java 1.1.1 contains built in support for compression and decompression, we implemented the compression and decompression modules as Java classes used in the application. The compression class that we implemented uses the zip format, as defined by PKWARE. The decompression class will extract files of this format from the compressed message file. In the absence of the Scenario Mail user agent, a receiver could still decompress a compressed message file for viewing by using a zip archive decompression program that supports long file names.

4.5. Scenario Presentation

The system uses an applet viewer to present (play) scenarios. Scenarios may be presented not only when messages are viewed, but also when the composer of a scenario would like to view the scenario before sending it in a message. For our implementation, we chose to use the Netscape Navigator 3.0 Gold browser to play the applet that presents the scenario to the user. By setting the AppletViewCmd and AppletViewCmdSuffix parameters in the ScenMail.cntl file, any Java enabled World Wide Web browser or other

applet viewing software may be used to present the scenarios.

4.6. Message Sending / Receiving

In our implementation, we also used Netscape Navigator to send and receive messages. Netscape Navigator allows third party developers to create add-on programs that can be integrated into the Navigator program to extend the capabilities of the browser. Netscape does this through the use of Plug Ins and Helper Applications. Netscape Navigator Plug Ins are dynamic link libraries (DLLs) that operate within the Web browser. A Netscape Navigator user can configure Netscape to call the DLL that implements the Plug In to view a designated MIME type.

A Helper application is an application that is launched when a designated MIME type is encountered. The Scenario Mail program is defined as a helper application that is launched to handle a body part that has MIME type ***application/x-JavaScenario***. Netscape also allows the association between file extensions and MIME types. By associating the extension for the compressed message file (".jsc") with the MIME type ***application/x-JavaScenario***, Netscape assigns the desired type to all compressed message files that are sent.

When Netscape Mail receives a message body part that has MIME type ***application/x-JavaScenario***, it invokes Scenario Mail to present the received scenario. In doing so, Netscape first saves the message file in a temporary directory. Netscape then automatically invokes Scenario Mail passing it the name of the compressed message file for the message to be presented. When Scenario Mail is invoked with a filename as a

command line parameter, it presents the message that is contained in the associated compressed message file.

4.7. Scenario Generation

One of the main functions of Scenario Mail is the generation of the Java applet code that implements a scenario. In each applet that is generated, some code is common to any scenario applet. To take advantage of this, the code generator uses a code template that consists of the applet code that is common to any scenario. The code generator then adds the scenario specific Java code when the scenario is specified. When a scenario has been fully specified, the scenario specific applet code is put together with the template code to form the actual applet. The text files that are used to form the template code are described in Figure 4.3. The contents of each of the template files is given in Appendix B.

File	File Contents
ScenImports.txt	Import statements for applet
ScenStatDeclare.txt	Declarations common to all scenarios
ScenInit.txt	*init* method, *action* method, and beginning of *run* method
ScenPaintBegin.txt	Beginning of *paint* method
ScenEnd.txt	End of the code for the scenario

Figure 4.3. Applet Template Files

In addition to generating the applet code, the program generates the html Web page that embeds the applet. The program also uses a template for this procedure. The applet class name (name of the compiled applet) and the size of the applet are added to complete the template when the Web page is generated. Since there is a small amount of template code needed for this task, the template code is contained in the application program, rather than in separate files.

4.8. File Naming Convention

To simplify implementation and increase user friendliness, the Scenario Mail program uses a file naming convention where the user does not need to supply the necessary extension for a filename when referring to the name of a scenario. The program automatically determines the extension for the scenario or message based on the context that the filename is given in. When a user saves a message as *FILENAME* (supplying no extension), then the files associated with the message are given the names shown in Figure 4.4:

File	Name
Applet Source Code	*FILENAME*.java
Compiled Applet Code	*FILENAME*.class
File List File	*FILENAME*.lst
Web Page That Embeds Applet	*FILENAME*.html
Compressed Message File	*FILENAME*.jsc

Figure 4.4. Assigned Filenames

CHAPTER 5

RESULTS

In order to provide a user friendly user interface, we implemented the interface to the system using a GUI windowing system. This interface allows the user to easily interact with the system to perform the main messaging functions such as composition and presentation of messages.

The main window of our multimedia mail system drives all main messaging functions. The main window contains three menus as shown in Figure 5.1.

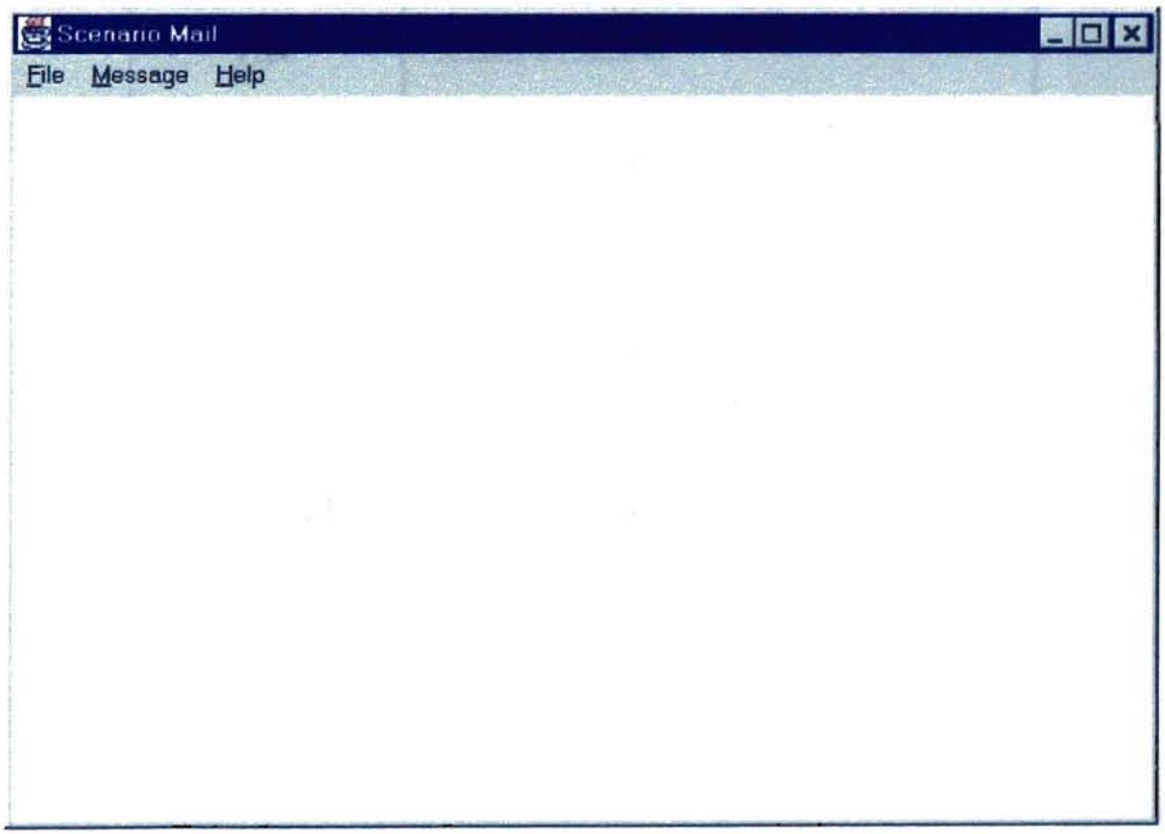

Figure 5.1. Scenario Mail Main Window

The **File** menu contains an **Exit** option that can be used to stop the execution of the application. The other main functions for the service are invoked through the **Message** menu. The main functions of the scenario service may be started by selecting the appropriate menu option under the **Message** menu. The options available under the **Message** menu are listed in Figure 5.2 below along with a description of the operations invoked through each option.

Message Menu Option	Operation Invoked
New	Begins creation of a new scenario.
View	Presents a scenario.
Save	Saves a scenario and creates the compressed message file to be sent as the message.
Send	Invokes the mail sending program for the sending of the created message.
Delete	Deletes all files that are associated with the specified message except for the media files used in the message.
Delete All	Deletes all files that are associated with the specified message including all media files used in the message.

Figure 5.2. Message Menu Options

5.1. Scenario Composition

The message creation process begins with the composition of a scenario. The scenario is specified by the message composer using the GUI interface. As the user creates the message, the Java applet code that implements the scenario will be generated. The Compose Message window shown in Figure 5.3 drives the scenario creation process. When creating each part of a scenario, the user has the option of creating a new slide or of adding a new media part or wait time that will be a part of the current non-slide part of a message.

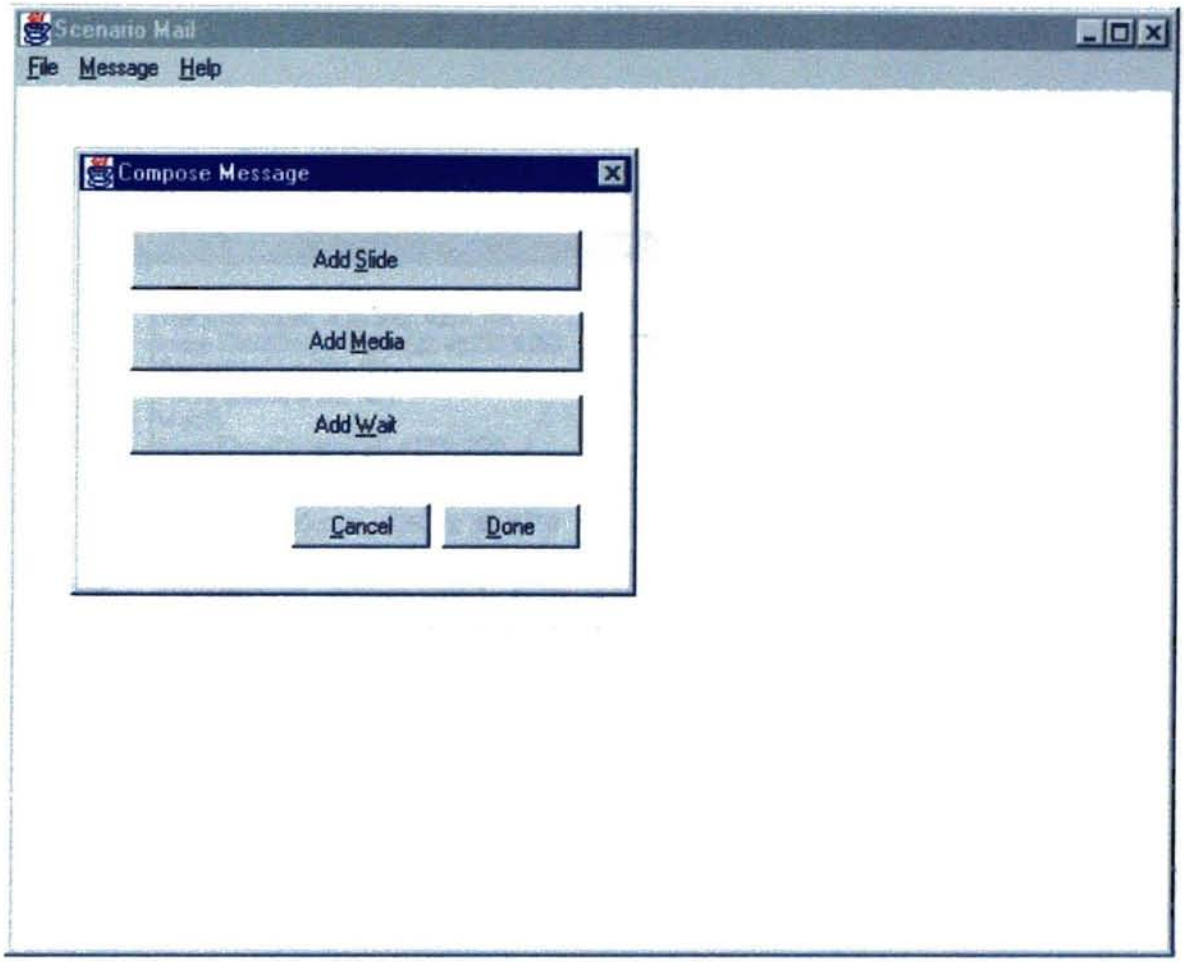

Figure 5.3. Composing a Message

5.1.1. Slide Composition

When creating a slide for a scenario, the Compose Slide window allows the user to specify the desired scenario to be played out for the slide currently being created. The part of the scenario that has been specified for the slide being composed is displayed in the scrolling box as shown in Figure 5.4.

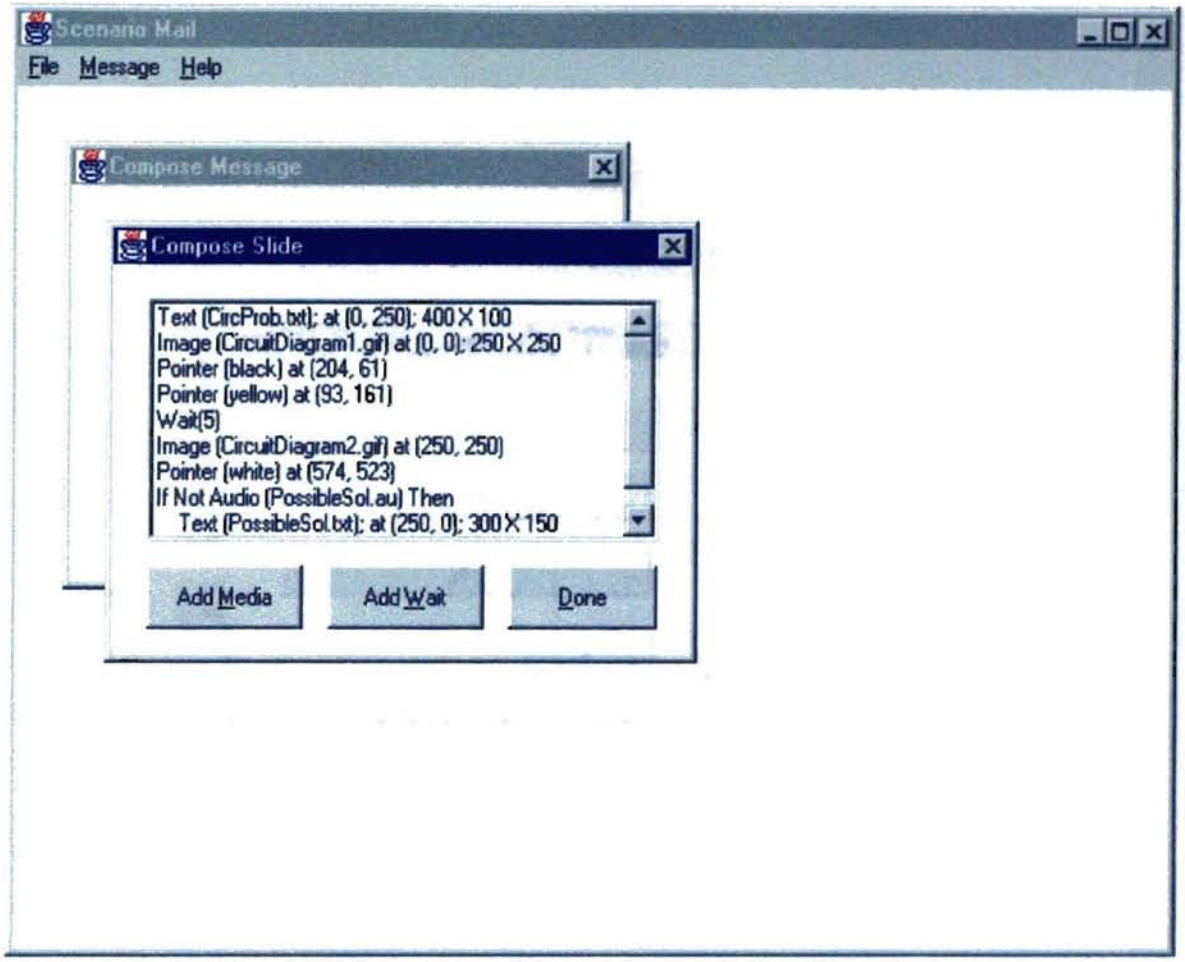

Figure 5.4. Composing a Slide

The Add Media button allows the user to add the next media part to be presented on the current slide or the current non-slide. The Add Media window shown in Figure 5.5 allows the user to select the type of media for the part to include. In our implementation we included support for image, audio, and text media.

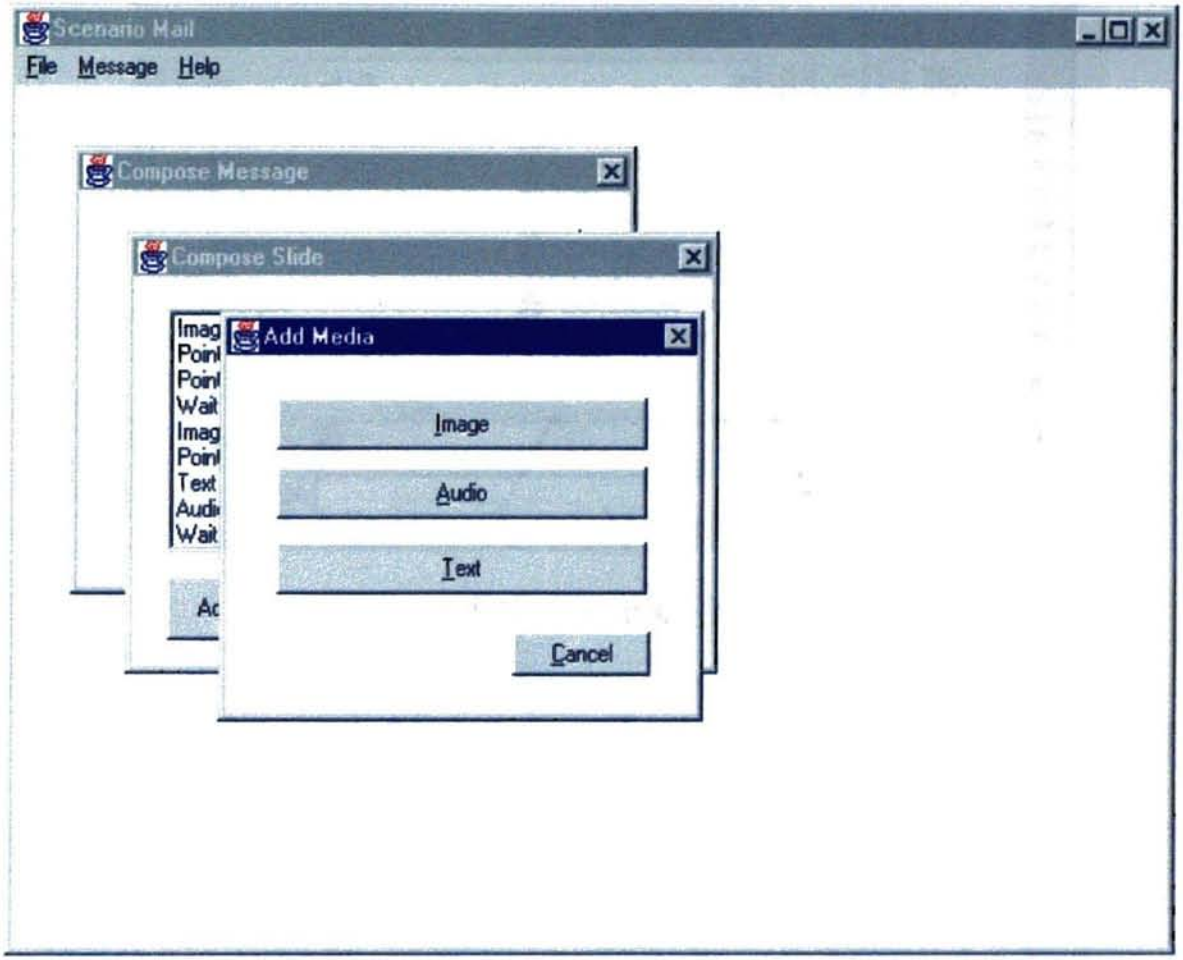

Figure 5.5. Adding Media

When adding a basic media part, the user must enter the name of the file containing the desired media part. To promote user friendliness, the service uses a file dialog window such as the one shown in Figure 5.6 to allow a user to browse and visually select a file of interest in cases when a filename must be given. This frees the user from having to remember the names of all of the files that they may want to include in a scenario.

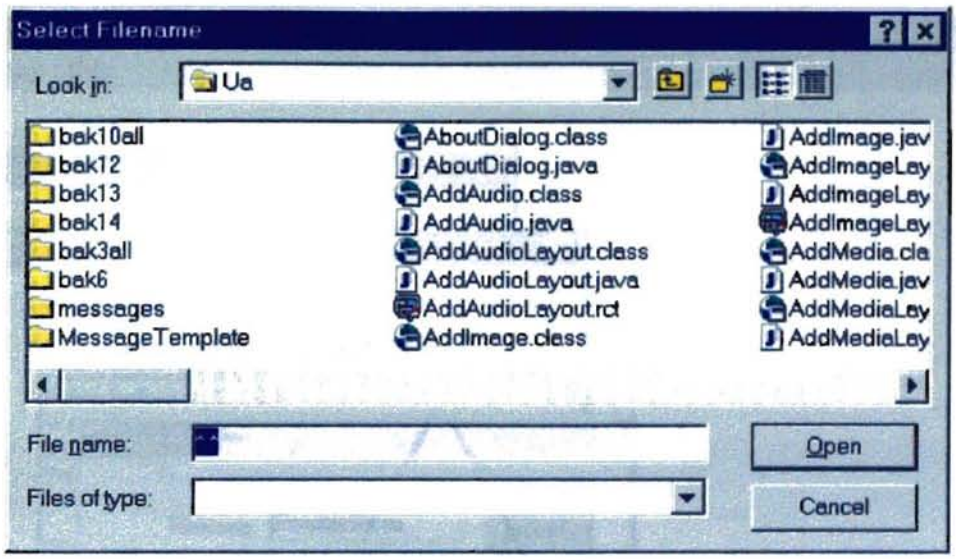

Figure 5.6. Filename Browsing

5.1.2. Audio Inclusion

For specification of an audio segment to be included in a scenario, selecting the filename containing the audio segment is all that is required. The Add Audio window shown in Figure 5.7 is used to get this selection.

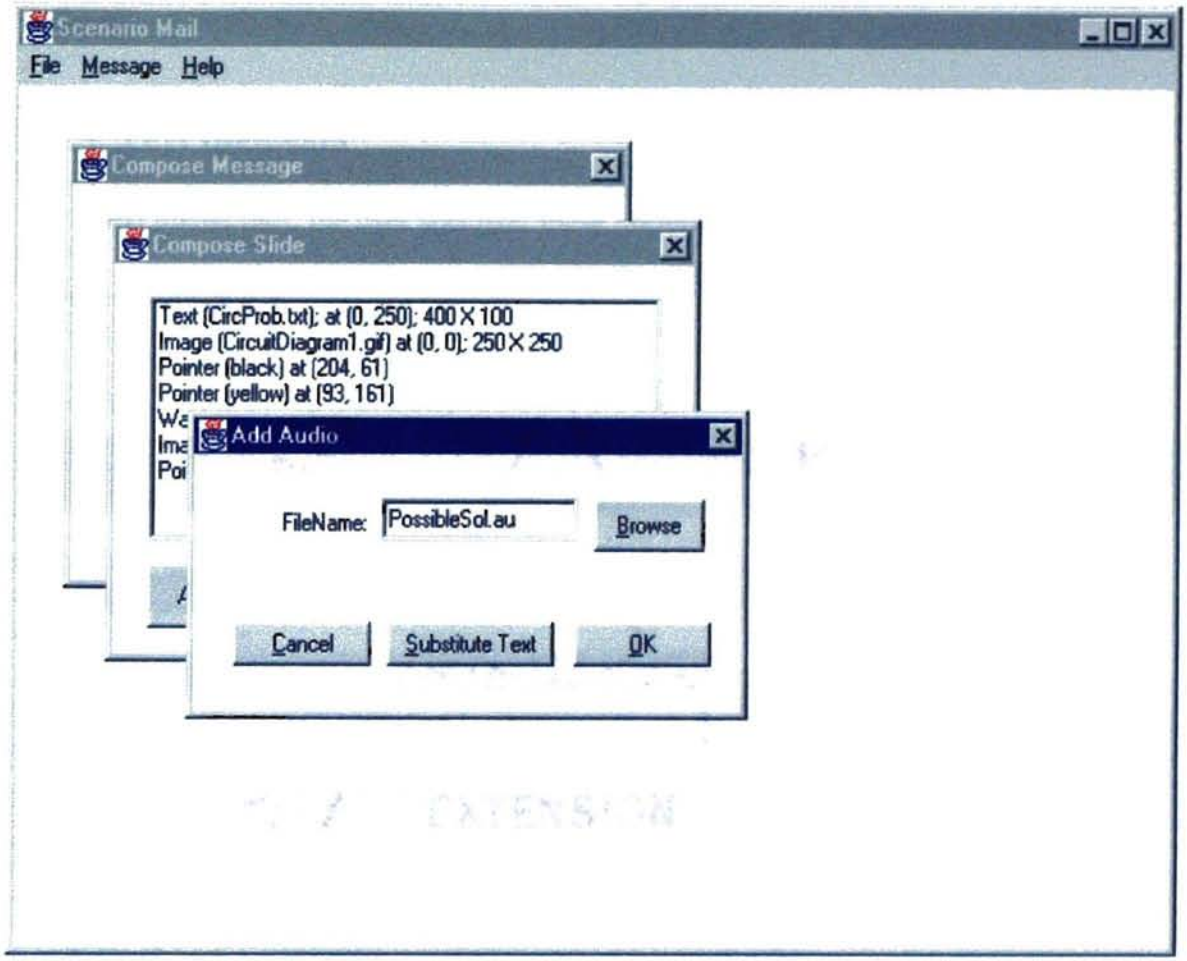

Figure 5.7. Adding Audio

When adding an audio clip to the scenario, a composer may specify a text segment that can be substituted for the audio if a viewer does not have audio capability on their machine. A composer may do this after entering the name of the audio file by selecting the **Substitute Text** button. The user could then specify the text segment to use as a substitute in the Add Text window shown in Figure 5.8.

In order for scenarios to be flexible, a scenario service must allow a composer to have total control over the size and placement of visible media. The composer of a scenario in the implemented service may specify both the placement location and size of

an image or text box.

5.1.3. Text Inclusion

For a text media part, the user specifies these preferences using the Add Text window shown in Figure 5.8.

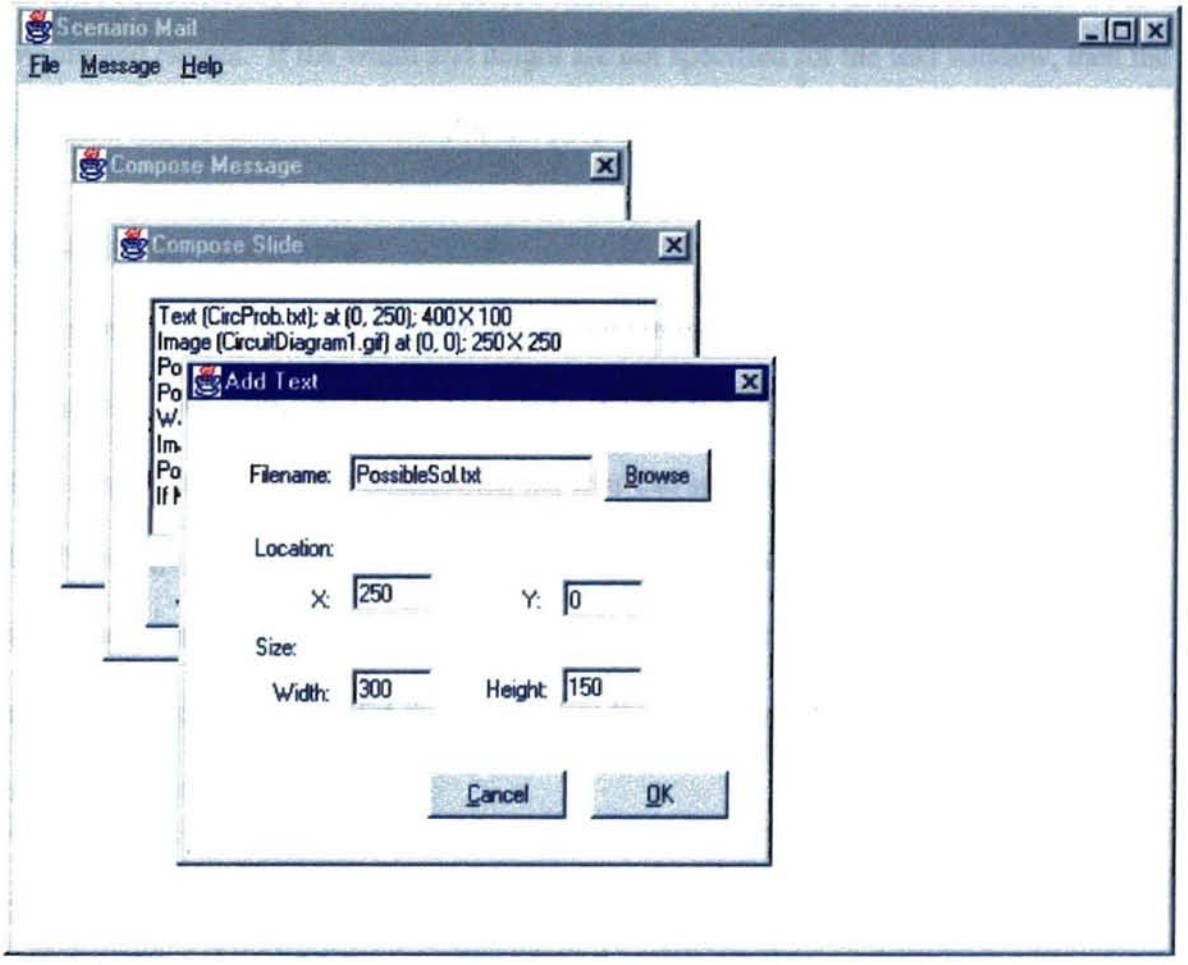

Figure 5.8. Adding Text

The display location for a visual media type is specified by giving the (X,Y) pixel values for the upper left corner of the window that will contain the presented media. The

location values are measured such that the upper left corner of applet window is location (0,0). Y values are specified such that the larger a Y value is for a point, the farther down in the applet window the point will be placed. The composer enters these location values in the text boxes labeled X and Y. If the composer does not specify the X and Y values, then the program assumes a location of (0,0). The user may also specify the width and height of the text window by entering these values in the text boxes labeled Width and Height. If the width and height are not specified for the text window, then the program will assume a default width and height.

5.1.4. Image Inclusion

The size and placement of an image is specified in the same manner on the Add Image Window shown in Figure 5.9. If the composer wishes to display an image at a size other than the normal size of the image, then the user can specify the desired height and width for the image. If they do not wish to alter the height and width of the image, then the composer must leave the Height and Width text boxes blank. If the composer does not specify a height and width for the image, then the image will be displayed at the normal size for the image.

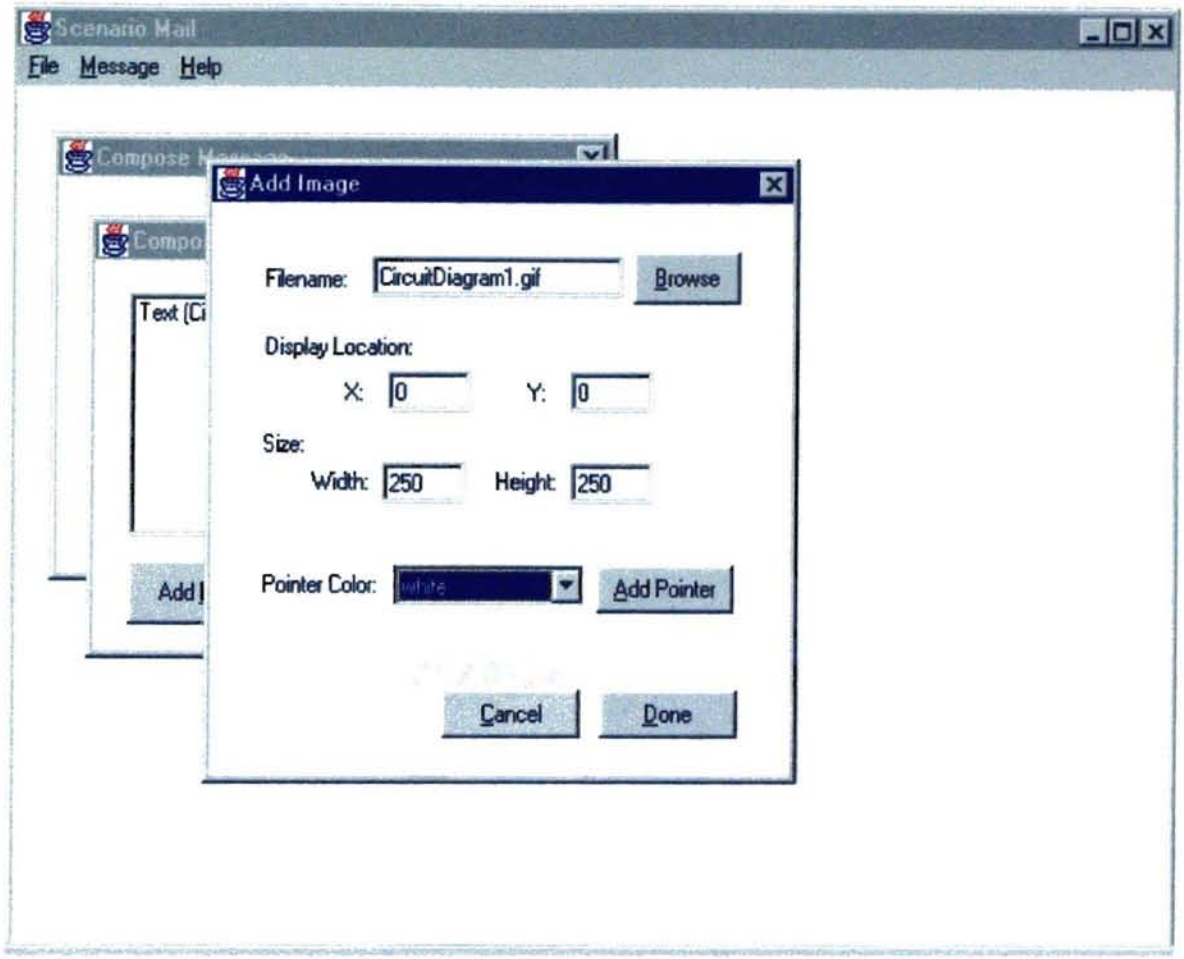

Figure 5.9. Adding an Image

In order to be able to use distinguishable pointers to facilitate a discussion, colored pointers are supported. The drop down box (Figure 5.10) on the Add Image window is used to specify the color for a pointer being added to an image.

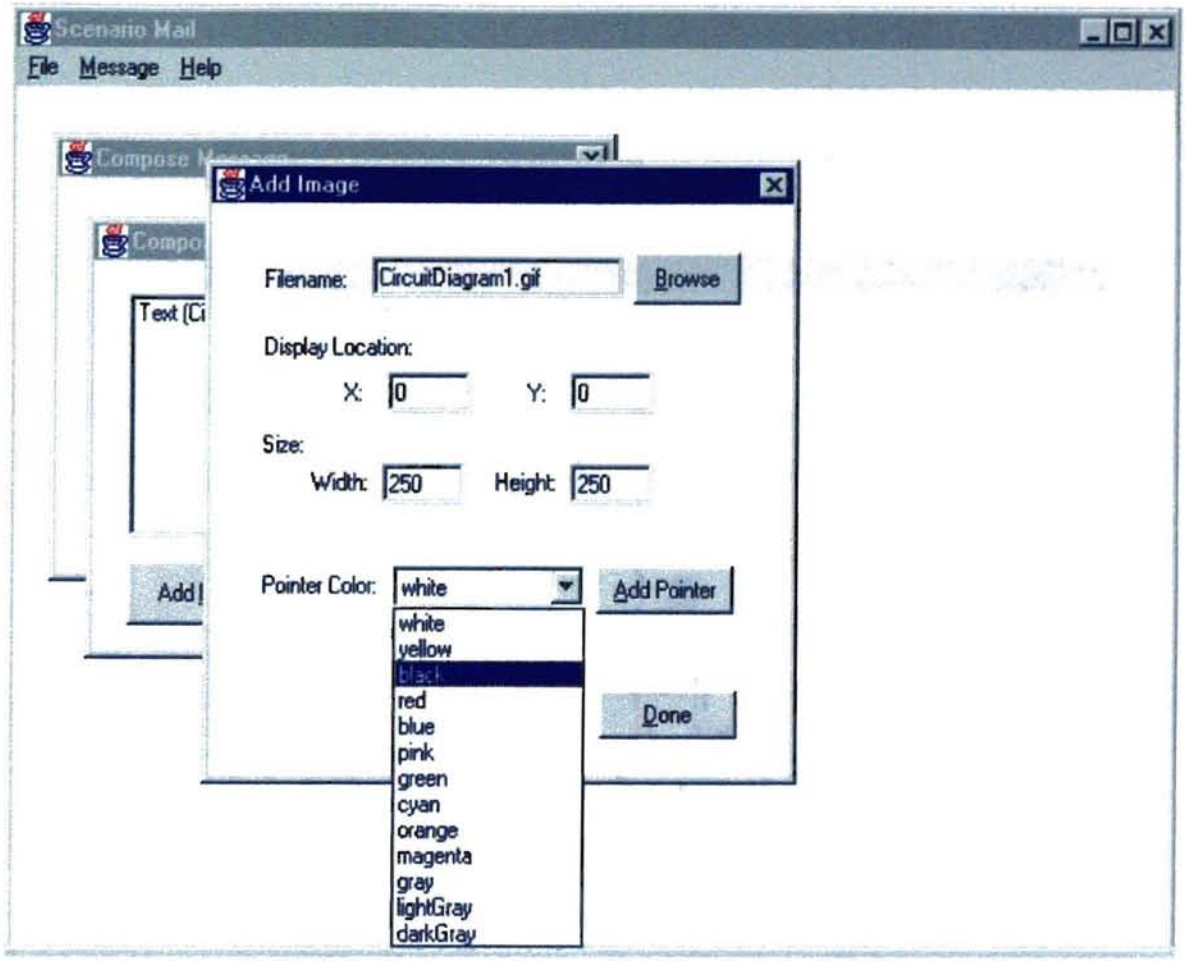

Figure 5.10. Selecting a Pointer Color

The actual placement of the pointer is specified on the Click to Add Pointer window which is shown in Figure 5.11. This window contains the image that the pointer will be placed on during presentation. This window will show the image at the size specified by the scenario composer. In order to specify the location to place the pointer on the image, the user must click on the desired location on the image with the left mouse button. This will cause the Java code to display the pointer to be generated.

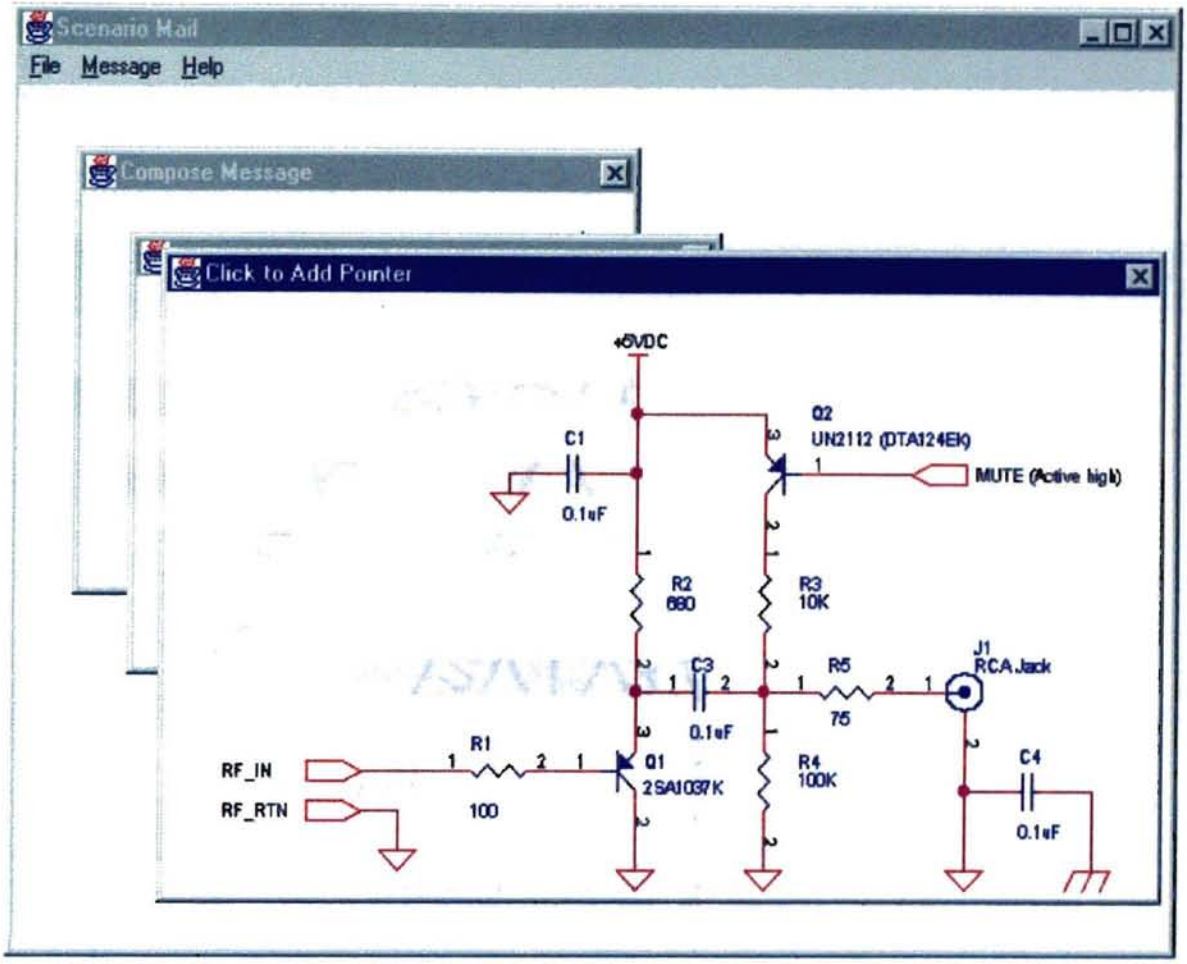

Figure 5.11. Adding a Pointer to an Image

5.1.5. Timing and Ordering

The order in which media parts are presented when a scenario is played is determined by the order in which the composer specifies media parts. Media parts within a slide or a "non-slide" part of a message can be presented in sequential order or in parallel. Unless the composer indicates otherwise, all such media parts will be played out in parallel. To cause parts of a message to be presented sequentially, the composer must either specify that the parts will belong to different slides or specify a wait time between the parts to be

presented sequentially. The wait time to pause the scenario is specified using the Add Wait window shown in Figure 5.12.

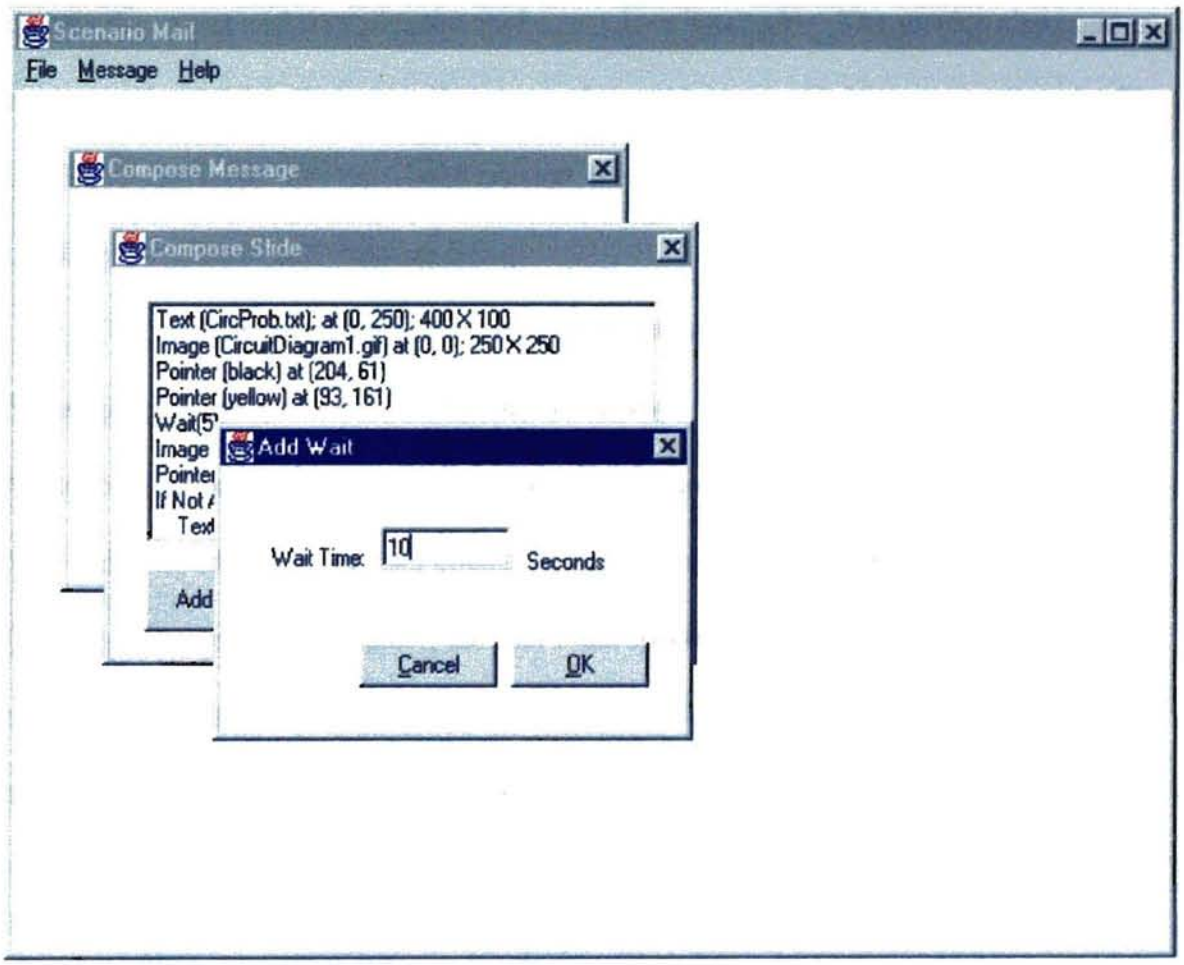

Figure 5.12. Adding Wait Time

If the composer wishes to present two media parts sequentially without a pause in between, the composer can specify a wait time of 0.

5.2. Message Sending

Before a message can be sent, it must be saved. After the entire scenario has been

specified, the composer must save the message by selecting the **Save** option from the **Message** menu on the Scenario Mail main window. After the composer enters the desired filename for the message (without an extension) in the file dialog box, the service creates the applet and compiles the applet into Java bytecodes. The service also creates the html World Wide Web page that the applet will be embedded into, as well as the file list file. After these files have been created, the program compresses the scenario files, creating the compressed message file.

To send the created message, the user must select the Send option from the **Message** menu on the Scenario Mail main window. Scenario Mail will then invoke Netscape Mail so that the Netscape - [Message Composition] window shown in Figure 5.13 is displayed. After the user enters the address information in the Netscape - [Message Composition] window, the user must specify to include the compressed message file as an attachment. The user can then send the message by selecting the Send button on the Netscape - [Message Composition] window.

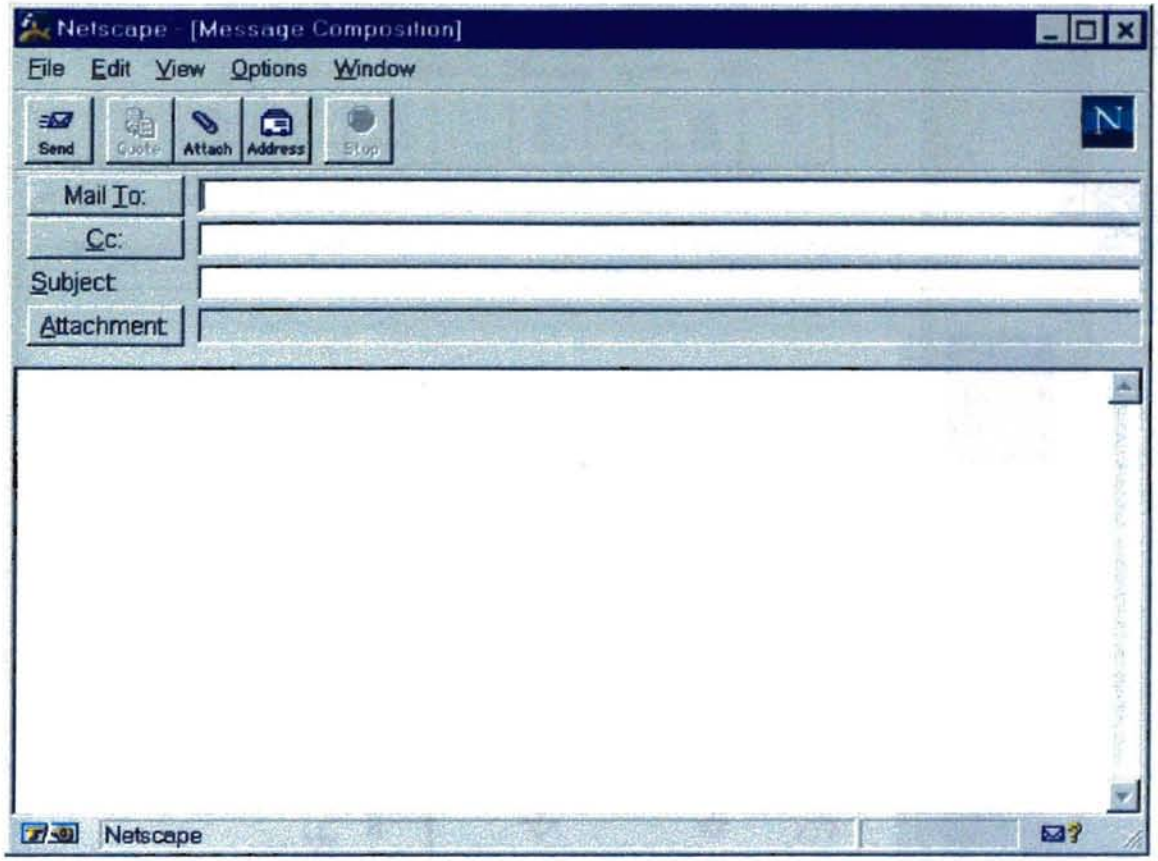

Figure 5.13. Sending the Message

5.3. Message Presentation

When a message is received, the receiving mail program will invoke Scenario Mail to automatically present the received message. To present a received message, Scenario Mail decompresses the compressed message file, extracting the original message files. Scenario Mail then invokes Netscape passing in the associated html file as a command line parameter. This causes the embedded applet to be executed inside the Web browser (Figure 5.14) to present the scenario for the user to view.

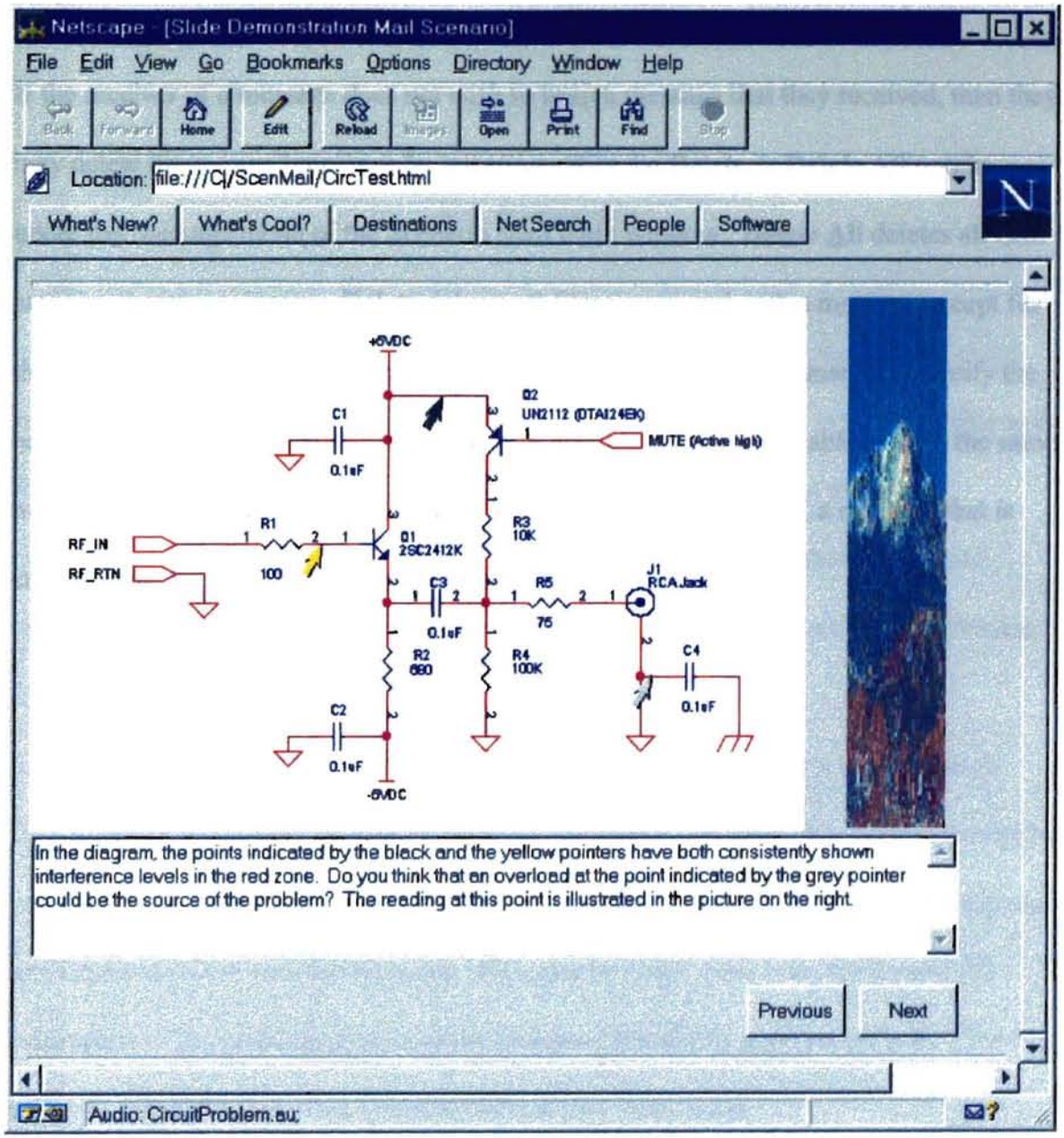

Figure 5.14. Viewing a Scenario

The **View** option under the **Message** menu of the Scenario Mail main window allows the receiver of a message to view a received scenario after the initial viewing. This option also allows the composer of a scenario to view the scenario before it is sent.

5.4. Message Deletion

If the receiver of a message does not wish to keep a message that they received, then they may delete the received message by selecting either the **Delete** or **Delete All** option under the Message menu of the Scenario Mail main window. **Delete All** deletes all files associated with a message. **Delete** deletes all files associated with a message except for the media files. A file dialog box will then be displayed where the user may specify the name of the message to delete. In order for a message creator to be able to send the same message more than once without having to re-compose the scenario, a message that is created remains on disk until the composer deletes the message.

CHAPTER 6

CONCLUSION

6.1. Summary

In this thesis, we proposed a new scheme for a multimedia scenario messaging system, and we presented an implementation of such a system. The scheme that we presented provides a more effective multimedia collaboration mechanism that is both portable and flexible. The addition of slide show capability allows more effective message presentation by allowing viewer interaction to drive movement between designated message parts. The addition of reference pointer capability also increases effectiveness by providing a visual reference tool for communication.

The implementation of the new scheme that we presented uses a user friendly GUI interface that allows operations such as message composition or message viewing to be performed easily. Since use of the World Wide Web continues to grow, we integrated our implementation with the Netscape Navigator browser. However, the control file parameters of the proposed system allow increased portability and flexibility by allowing a user to specify the interface components that they wish to use.

6.2. Future Work

The implementation provides an effective multimedia messaging tool that is portable and flexible. However, there are some other features that may be added in the future to build on our work. These features are listed below:

- Ability to change an existing scenario or partial scenario within Scenario Mail.
- Addition of support for other media, such as video, to the system.
- A method of composing scenario slides where the user can use mouse clicks and mouse drags to visually specify the layout of a slide.
- Automatic attachment of the compressed message file to a message being sent.
- Media capturing capability integrated into the service that allows media file creation at compose time.

REFERENCES

[Aga95] Tahar Agastani. Multimedia Mail: An Implementation Providing a Scenario Service. Published Masters Thesis, Computer Science Department, Oklahoma State University, Stillwater, OK, 1995.

[AH+95] Sudhir R. Ahuja, H. Fred Haisch, Ram S. Ramamurthy, and Lucinda M. Sanders. Multimedia Collaboration. AT&T Technical Journal, Vol. 74, No. 5, September/October 1995, pp. 46-53.

[BT91] Nathaniel S. Borenstein and Chris A. Thyberg. Power, Ease of Use and Cooperative Work in a Practical Multimedia Message System. International Journal of Man-Machine Studies, Vol. 34, 1991, pp. 23-31.

[Bor91] Nathaniel Borenstein. Multimedia Electronic Mail: Will the Dream Become a Reality?. Communications of the ACM, Vol. 34, No. 4, April 1991, pp. 117-119.

[Bor92] Nathaniel Borenstein. Computational Mail as Network Infrastructure for Computer-Supported Cooperative Work. Bellcore, ftp://thumper.bellcore.com/pub/nsb/CSW-ATOMICMAIL.txt.

[BR93] Nathaniel S. Borenstein and Marshall T. Rose. MIME Extenstions for Mail-Enabled Applications: application/Safe-Tcl and multipart/enabled-mail, Working Draft, http://minsky.med.virginia.edu/sdm7g/Projects/Python/safe-tcl/safte-tcl.txt, November, 1993.

[Bor94] Nathaniel S. Borenstein. Email With a Mind of Its Own: The Safe-Tcl Language for Enabled Mail. A Copy of a paper from the ULPAA '94 Conference in Vancouver, http://wuarchive.wustl.edu/packages/first-virtual/nsb/, May, 1994.

[BF96a] Nathaniel Borenstein and Ned Freed. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. working draft, Internet Engineering Task Force, http://www.ietf.cnri.reston.va.us/ids.by.wg/822ext.html, March, 1996.

[BF96b] Nathaniel Borenstein and Ned Freed. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. working draft, Internet Engineering Task Force, http://www.ietf.cnri.reston.va.us/ids.by.wg/822ext.html, March 1996.

[BF96c] Nathaniel Borenstein and Ned Freed. Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples. working draft, Internet Engineering Task Force, http://www.ietf.cnri.reston.va.us/ids.by.wg/822ext.html, March 1996.

[CG95] Sylvain Carrier and Nicolas D. Georganas. Practical Multimedia Electronic Mail on X.400. IEEE Multimedia, Vol. 2, No. 4, Winter 1995, pp. 12-23.

[CH96] Gary Cornell and Cay S. Horstmann. *Core Java*. SunSoft Press, 1996.

[CMC95] Nancy Cox, Charles T. Manley, Jr. and Francis E. Chea. *Lan Times Guide to Multimedia Networking*. McGraw-Hill, Inc., 1995.

[Cro82] David H. Crocker. Standard for the Format of ARPA Internet Text Messages, STD 11, RFC 822, UDEL, http://www.wu-wien.ac.at:8082/rfc/rfc822.hyx/$$root, August 13, 1982.

[DM95] Per Danvind and Mattias Mattsson. Computational Mail. Masters Thesis in Computer Science and Engineering, University of Lulea, Sweden, http://www.cdt.luth.se/~mattias/ex_jobb/report/MasterThesis.html, November, 24, 1995.

[GT95] Simon J. Gibbs and Dionysios C. Tsichritzis. *Multimedia Programming: Objects, Environments and Frameworks*. ACM Press, 1995.

[GJ94] Stephen J. Griesmer and Richard W. Jesmajian. Evolution of Messaging Standards. AT&T Technical Journal, Vol. 73, No. 3, May/June 1994, pp. 21-45.

[HK94] W. Herzner and F. Kappe. *Multimedia/Hypermedia in Open Distributed Environments*. Springer-Verlag, 1994.

[KE96] Ahmed Karmouch and James Emery. A Playback Schedule Model for Multimedia Documents. IEEE Multimedia, Vol. 3, No. 1, Spring 1996, pp. 50-61.

[KGH94] B Kervella, V Gay, and E Horlait. Integration of a Scenario Service in a Multimedia Messaging System. In *Proceedings of the IASTED/ISMM: Distributed Multimedia Systems and Applications*, Honolulu, Hawaii, August 1994, pp. 119-122.

[PT96] Maria Jose Perez-Luque and Thomas D. C Little. A Temporal Reference Framework for Multimedia Synchronization. IEEE Journal on Selected Areas in Communications, Vol. 14, No. 1, January 1996, pp. 36-51.

[RP+96] S. V. Raghavan, B. Prabhakaran, and Satish K. Tripathi. Synchronization Representation and Traffic Source Modeling in Orchestrated Presentation. IEEE Journal on Selected Areas in Communications, Vol. 14, No. 1, January 1996, pp. 104-113.

[RB94] Marshall T. Rose and Nathaniel Borenstein. A Model for Enabled Mail (EM), Working Draft, Bellcore, Dover Beach Consulting, Inc., http://hcirisc.cs.binghamton.edu/~mrwizard/enabled.txt, July 19, 1994.

[SKD96] James Schnepf, Joseph A. Konstan, and David Hung-Chang Du. Doing FLIPS: FLexible Interactive Presentation Synchronization. IEEE Journal on Selected Areas in Communications, Vol. 14, No. 1, January 1996, pp. 114-125.

[SD+96] Patrick Senac, Michel Diaz, Alain Leger, and Pierre de Saqui-Sannes. Modeling Logical and Temporal Synchronization in Hypermedia Systems. IEEE Journal on Selected Areas in Communications, Vol. 14, No. 1, January 1996, pp. 84-103.

[Szu95] Bohdan O. Szuprowicz. *Multimedia Networking*. McGraw-Hill, Inc., 1995.

[Wal95] Rob Walters. *Computer-Mediated Communications: Multimedia Applications*. Artech House, Inc., 1995.

[WKD96] Johnny Wong, Sriram Kini and Kishore Doobagunta. Synchronization in Specification-based Multimedia Presentations. Software-Practice and Experience, Vol. 26, No. 1, January 1996, pp. 71-81.

APPENDIX A

Scenario Mail Code

The source code for the Scenario Mail application is listed below. This listing does not include the source code generated by Microsoft Visual J++ that was used in the application..

ScenMail Class

```
import java.awt.*;
import java.io.*;
import java.lang.*;
import java.util.*;
import ComposeMessage;
import StrList;
import ZipFileCompress;
import ZipFileDecompress;

public class ScenMail extends Frame implements Runnable
//---------------------------------------------------------------------------
// Class ScenMail
// Description:
// Implements the main window for scenario mail and drives the
// functions for creating scenarios, viewing scenarios,
// saving messages, sending messages, and deleting messages.
//---------------------------------------------------------------------------
{   // class constants:
   private static int NONE = 1;                  // Part of message being created is slide
   public static int SLIDE = 2;                  // Part being created not part of slide
   public static int NON_SLIDE = 3;              // Part being created is undetermined
   public static int ALL = 4;                    // Deletion with media files deleted
   public static int NOT_ALL = 5;                // Deletion with media files not deleted
   static final int IMAGE = 6;                   // Indicates image media type
   static final int AUDIO = 7;                   // Indicates audio media type
   static final int TEXT = 8;                    // Indicates text media type
   public static String LIST_FILE_EXT = new String("lst");
   public static String CONTROL_FILE = new String("ScenMail.cntl");
   private static int MaxFiles = 100;            // Maximum # of files in one message
   public static String INSTALL_PATH =    new String("C:\\ScenMail\\");
   // class variables:
   private String CompileCmd = null;             // Command to invoke java
   private String CompileCmdSuffix=null;         // Suffix of Command to invoke compiler
   private String AppletViewCmd = null;          // Command to invoke applet viewer
   private String AppletViewCmdSuffix = null; // Suffix of Command to invoke  viewer
   private String MailSendCmd = null;            // Command to invoke mail sender
   private String MailSendCmdSuffix = null;  // Suffix of Command to invoke sender
```

```
private String PlayerPathname = null;      // Path of applet viewer
private int AppletWidth;                   // Width of applet in pixels
private int AppletHeight;                  // Height of applet in pixels
private int ApplicationWidth;              // Width of main frame window in pixels
private int ApplicationHeight;             // Height of main frame window in pixels
private int PointerSize;                   // scaling factor to determine pointer size
private String MESSAGE_EXT = new String("jsc");
private int MessageExtLen;                 // Length of extension for message filename
private boolean DeleteSource=true;         // If true, remove source code, when deleting
private ComposeMessage ComposeBox;         // Message creation form
private MsgBox msg;                        // Form to display messages
private int state;                         // Current part of message being created
private int ImageState;                    // Indicates part of state consisting of images
                                           // before next wait
private boolean ImageStateBegan;           // True if image has been added after
                                           // latest wait in current slide or non-slide
private int SlideNum;                      // Number of slide in creation sequence
private int ImageNum;                      // Number of image in inclusion sequence
private int AudioNum;                      // # of audio segment in inclusion sequence
private StrList FileList;                  // List of files used in the message
private boolean ScenStarted = false;       // Message is not being created yet
private int PartCreating;                  // Part of message currently being created
private boolean FilesOpen;                 // True if files used to store scenario open
private String ScenFilename = null;        // File scenario is stored in
private FileOutputStream ScenRunStream;// Portion of scenario code that
private PrintStream ScenRunFile;           // --> contains run method of applet
private FileOutputStream ScenPaintStream; // Portion of scenario code that
private PrintStream ScenPaintFile;         // --> contains paint method of applet
private FileOutputStream ScenDeclareStream; // Portion of scenario code that
private PrintStream ScenDeclareFile;       // --> contains declarations for applet
private boolean MODAL = false;
public Thread ScenarioThread = null;

public ScenMail()
//************************************************************************
//* Create title and menus for main frame window
//************************************************************************
{
  setTitle("Scenario Mail");

  // Add the menu options to the ScenMail frame:
  MenuBar mbar = new MenuBar();
  // Set upFile menu:
  Menu m = new Menu("File");
  m.add(new MenuItem("Exit"));
```

```
    mbar.add(m);
    // Set Up Message menu:
    m = new Menu("Message");
    m.add(new MenuItem("New"));
    m.add(new MenuItem("View"));
    m.add(new MenuItem("Save"));
    m.add(new MenuItem("Send"));
    m.add(new MenuItem("Delete"));
    m.add(new MenuItem("Delete All"));
    mbar.add(m);
    // Set up Help menu:
    m = new Menu("Help");
    m.add(new MenuItem("About"));
    mbar.add(m);
    setMenuBar(mbar);
    GetControlVals();

    if (ScenarioThread == null)
    {
      ScenarioThread = new Thread(this);
      ScenarioThread.start();
    }
}

public void run()
{}

void GetControlVals()
//*************************************************************************
//* Get the value of each parameter from the control file and store the
//* value in the corresponding program variable.
//*************************************************************************
{
    DataInputStream ControlFile = null;  // control file
    boolean EndFile = false;     // true when at end of control file
    String CurLine = null;           // current line read from control file
    String ControlParam = null;       // parameter from control file
    String ParamValue = null;         // value of the parameter

    try          // open control file
    {
      ControlFile = new DataInputStream(new FileInputStream(INSTALL_PATH
                                        + CONTROL_FILE));
    }
    catch(IOException e)
```

```
{
   msg = new MsgBox(this, "ERROR", "ERROR - file '" + INSTALL_PATH
                          + CONTROL_FILE + "' Not Found", " ");
   msg.show();
}

// while not at end of control file, get get the current parameter
// and assign it's value to the corresponding program variable.
while (EndFile == false)
{
   try
   {
      CurLine = ControlFile.readLine();
      if (CurLine != null)
      {
         CurLine = CurLine.trim();
         ControlParam = GetWord(CurLine);

         if (ControlParam != null)
         {
            ParamValue = CurLine.substring(ControlParam.length()).trim();

            if ((ParamValue != null) && (ParamValue.length() > 0))
            {
               if (ControlParam.compareTo("CompileCmd") == 0)
                  CompileCmd = ParamValue;
               else if (ControlParam.compareTo("CompileCmdSuffix") == 0)
                  CompileCmdSuffix = ParamValue;
               else if (ControlParam.compareTo("AppletViewCmd") == 0)
                  AppletViewCmd = ParamValue;
               else if (ControlParam.compareTo("AppletViewCmdSuffix") == 0)
                  AppletViewCmdSuffix = ParamValue;
               else if (ControlParam.compareTo("MailSendCmd") == 0)
                  MailSendCmd = ParamValue;
               else if (ControlParam.compareTo("MailSendCmdSuffix") == 0)
                  MailSendCmdSuffix = ParamValue;
               else if (ControlParam.compareTo("AppletWidth") == 0)
                  AppletWidth = Atoi(ParamValue);
               else if (ControlParam.compareTo("AppletHeight") == 0)
                  AppletHeight = Atoi(ParamValue);
               else if (ControlParam.compareTo("ApplicationWidth") == 0)
                  ApplicationWidth = Atoi(ParamValue);
               else if (ControlParam.compareTo("ApplicationHeight") == 0)
                  ApplicationHeight = Atoi(ParamValue);
               else if (ControlParam.compareTo("PointerSize") == 0)
```

```
                    PointerSize = Atoi(ParamValue);
                else
                {
                    msg = new MsgBox(this, "WARNING",
                            "Unknown Parameter In Control file: ", ControlParam);
                    msg.show();
                }
            }
        }
        }
        else
            EndFile = true;
    }
    catch(IOException e)
    {
        msg = new MsgBox(this, "ERROR", "ERROR " + e, " ");
        msg.show();
    }
}

try
{
    ControlFile.close();
}
catch(IOException e)
{
    msg = new MsgBox(this, "ERROR", "ERROR - Can Not Close File: " +
                        ScenFilename + ".txt", " ");
    msg.show();
}

if (MESSAGE_EXT == null)
    MessageExtLen = 0;
else
    MessageExtLen = MESSAGE_EXT.length();
}

String GetWord(String str)
//**************************************************************
//* Return the first word in a string
//*
//* Parameters:
//*      str: string to get the first word of
//**************************************************************
{
```

```
    int i = 0;                  // position in string
    boolean WordStored = false;   // true when word is stored
    String Word = new String(""); // current part of first word

    if (str != null)
      while ((WordStored == false) && (i < str.length()))
      {
        if (str.charAt(i) == ' ')
          WordStored = true;
        else
        {
          Word = Word + str.charAt(i);
          i++;
        }
      }

    return Word.trim();
}

public void InitMessage()
//*************************************************************************
//* Prepare the necessary variables and files for creation of new message
//*************************************************************************
{
    state = 0;
    ImageState = 0;
    ImageStateBegan = true;
    SlideNum = 0;
    ImageNum = 0;
    AudioNum = 0;
    PartCreating = NONE;
    ScenFilename = null;
    FileList = new StrList(this, MaxFiles);

    // open file that will contain the code added to the run method of the applet
    try
    {
      ScenRunStream = new FileOutputStream(INSTALL_PATH
                                           + "ScenRunEnd.txt");
      ScenRunFile = new PrintStream(ScenRunStream);
    }
    catch(IOException e)
    {
      msg = new MsgBox(this, "ERROR", "Error - Can Not Open File: ",
                       "ScenRunEnd.txt");
```

```
      msg.show();
   }

   // open the file that will contain the code added to the paint method
   // of the applet
   try
   {
      ScenPaintStream = new FileOutputStream(INSTALL_PATH +
                                                "ScenPaintEnd.txt");
      ScenPaintFile = new PrintStream(ScenPaintStream);
   }
   catch(IOException e)
   {
      msg = new MsgBox(this, "ERROR", "Error - Can Not Open File: ",
                              "ScenPaintEnd.txt");
      msg.show();
   }

   // open the file that will contain the declarations added to the applet
   try
   {
      ScenDeclareStream = new FileOutputStream(INSTALL_PATH +
                                                "ScenDeclare.txt");
      ScenDeclareFile = new PrintStream(ScenDeclareStream);
   }
   catch(IOException e)
   {
      msg = new MsgBox(this, "ERROR", "Error - Can Not Open File: ",
                              "ScenDeclare.txt");
      msg.show();
   }

   FilesOpen = true;
   ScenStarted = true;
}

public int GetNameLen(String str)
//*****************************************************************
// Return the length of the string str up to a delimiting char.
//
// Parameters:
//      str: string to find the length of
//*****************************************************************
{
   int len = 0;                  // name length
```

```
int i;                    // current char. in string
int LenWithoutAsk = 0;    // length of string without "*.*" appended
boolean ExtTraversed = false;   // true if extension has been passed
boolean EndsInAsk;        // true if string ends in an asterisk

if (str != null)
{
   i = str.length() - 1;

   // Since the filename returned by a file dialog box in Java, often
   // contains "*.*" appended to the end of it, the "*.*" must also
   // be removed to extract the filename.
   if (str.charAt(i) == '*')
      EndsInAsk = true;
   else
      EndsInAsk = false;

   // determine the number of characters following the filename that
   // must be discarded
   while ((i >= 0) && (ExtTraversed == false))
   {
      if ((str.charAt(i) == '.') && (str.charAt(i - 1) != '*'))
         if (EndsInAsk == true)
         {
         // have just moved past the "*.*", we must now continue
         // in order to move past the filename extension
            EndsInAsk = false;
            LenWithoutAsk = i;   // if no extension before "*.*",
                        // this will be length of filename
            i--;
         }
         else
            // encountered the end of the filename so stop
            ExtTraversed = true;
      else
         i--;
   }

   if (i < 0)
      // if the string contained a filename without an extension, but
      // with "*.*" appended, then return the length of the string
      // - the length of "*.*".
      if  (LenWithoutAsk > 0)
         return LenWithoutAsk;
      else
```

```
        // filename did not have an extension or "*.*" on the end
            return str.length();
        else
            return i;
    }
    else
        return 0;
}

public boolean action(Event evt, Object arg)
{
    FileDialog GetFilenameBox;    // dialog box used to get filename from user
    Process Proc;                 // variable representing the running process
    Runtime RTime;                // runtime environment of running process
    String FullAppletViewCmd = null;  // full command to play the applet
    String FName = null;

    if (evt.target instanceof MenuItem)
    {
        if(arg.equals("Exit"))
        // User selected Exit option from File menu
        // ==> Clean up and exit program.
        {
            CloseFiles(FilesOpen);
            FilesOpen = false;

            System.exit(0);
        }
        else if(arg.equals("Save"))
        // User selected Save option from Message menu.
        // ==> Save the Java applet that implements the scenario.
        {
            setCursor(Frame.WAIT_CURSOR);

            if (ScenStarted)
            {
                if (PartCreating != SLIDE)
                    AddEndStateCode(NON_SLIDE);

                PartCreating = NONE;
                FName = new String(SaveScenario());
                SaveMessage(FName);
            }

            setCursor(Frame.DEFAULT_CURSOR);
```

```
}

else if(arg.equals("New"))
// User selected New option from Message menu
// ==> Initialize and prepare for new message composition.
{
   InitMessage();
   ComposeBox = new ComposeMessage(this);
   ComposeBox.show();
}
else if(arg.equals("Send"))
// User selected Save option from Message menu
// ==> Ensure message has ben saved and invoke mail sending software
{
   try
   {
      RTime = Runtime.getRuntime();
      Proc = RTime.exec(MailSendCmd);
   }
   catch(IOException e)
   {
      msg = new MsgBox(this, "ERROR", "Error " + e, " ");
      msg.show();
   }
}
else if(arg.equals("Delete"))
// User selected View option from Message menu
// ==> Present (play) the message for the user.
{
   DeleteMessage(false);
}
else if(arg.equals("Delete All"))
// User selected View option from Message menu
// ==> Present (play) the message for the user.
{
   DeleteMessage(true);
}
else if(arg.equals("View"))
// User selected View option from Message menu
// ==> Present (play) the message for the user.
{
   // get the filename of the message to present.
   GetFilenameBox = new FileDialog(this, "Select Scenario Filename",
                                   FileDialog.LOAD);
   GetFilenameBox.setDirectory(".");
```

```
            if (ScenFilename != null)
               GetFilenameBox.setFile(ScenFilename);

            GetFilenameBox.show();
            ScenFilename = GetFilenameBox.getFile();
            String ScenFileDir = GetFilenameBox.getDirectory();

            if ((GetFilenameBox != null) && (ScenFilename != null))
            {
               ScenFilename = ScenFileDir + ScenFilename.substring(0,
                                 GetNameLen(ScenFilename));

               // Play the scenario applet selected by the user (ScenFilename)
               try
               {
                  FullAppletViewCmd = new String(AppletViewCmd + " " +
                                                      ScenFilename + ".html");

                  if (AppletViewCmdSuffix!= null)
                     FullAppletViewCmd = FullAppletViewCmd + " "
                                             + AppletViewCmdSuffix;

                  RTime = Runtime.getRuntime();
                  Proc = RTime.exec(FullAppletViewCmd);
               }
               catch(IOException e)
               {
                  msg = new MsgBox(this, "ERROR", "Error " + e, " ");
                  msg.show();
               }
            }
         }
      }
      else
         return false;

      return true;
   }

   public boolean handleEvent(Event evt)
   {
      if ((evt.id == Event.WINDOW_DESTROY) && (evt.target == this))
      // if "exit button" was clicked, then close open files and exit
      {
         CloseFiles(FilesOpen);
```

```
      FilesOpen = false;
      System.exit(0);
    }
    return super.handleEvent(evt);
}

public int GetPartCreating()
//*************************************************************************
// Return the type of message part that the user is currently creating
//*************************************************************************
{       return PartCreating;    }


public void SetPartCreating(int NewPartCreating)
//*************************************************************************
//* Store the type of message part that the user is currently creating
//*
//* Parameters:
//*       NewPartCreating: type of message part being created
//*************************************************************************
{       PartCreating = NewPartCreating;          }


public void SaveMessage(String filename)
//*************************************************************************
// Create the compressed message file containing the applet, html page,
// file list file, and all media files used in the message
//
// Parameters:
//      filename: name to save compressed message file as
//       (extension will be ".jsc" unless user overrides)
//*************************************************************************
{
   if (filename != null)
   {
      ZipFileCompress CompressedMessFile = new ZipFileCompress(this, filename
                                                + "." + MESSAGE_EXT);

      DataInputStream FListFile = null;   // file list file
      boolean EndFile = false;          // true if at end of file
      String CurFName = null;           // current filename from list

      filename = filename.substring(0, GetNameLen(filename));
      try
      {
```

```
            FListFile = new DataInputStream(new FileInputStream(INSTALL_PATH
                                                + filename + "." + LIST_FILE_EXT));
        }
        catch(IOException e)
        {
            msg = new MsgBox(this, "ERROR", "ERROR - file '" + filename + "."
                                + LIST_FILE_EXT + "' Not Found", " ");
            msg.show();
        }

        //* while not at end of TextFile, add the current character from the
        //* file to the text window and move to next char in file
        while (EndFile == false)
        {
            try
            {
                CurFName = FListFile.readLine();
                if (CurFName != null)
                {
                    CompressedMessFile.AddFile(CurFName);
                }
                else
                    EndFile = true;
            }
            catch(IOException e)
            {
                msg = new MsgBox(this, "ERROR", "ERROR " + e, " ");
                msg.show();
            }
        }

        try
        {
            FListFile.close();
        }
        catch(IOException e)
        {
            msg = new MsgBox(this, "ERROR", "ERROR - Can Not Close File: " +
                                ScenFilename  + ".txt", " ");
            msg.show();
        }

        CompressedMessFile.Save();
    }
}
```

```
public String SaveScenario()
//*************************************************************************
// Create the scenario, the html page, the
// Return the name of the file that the message is saved in.
//*************************************************************************
{
   Process Proc;              // process to execute compile
   Runtime RTime;             // current runtime
   String FullCompileCmd = null;  // command to compile applet

   // Get the name of the file to save the scenario as from the user
   FileDialog GetFilenameBox = new FileDialog(this, "Save as", FileDialog.SAVE);
   GetFilenameBox.setDirectory(".");
   GetFilenameBox.show();
   ScenFilename = GetFilenameBox.getFile();

// As long as valid filename was specified, create:
//  1) rest of scenario applet
//  2) html page applet will be embedded in
//  3) compile the applet
   if ((GetFilenameBox != null) && (ScenFilename != null))
   {
      ScenStarted = false;
      ScenFilename = ScenFilename.substring(0, GetNameLen(ScenFilename));
      // generate html page
      BuildWebPage(ScenFilename);
      // finish creation of applet
      AddNumStatesCode();
      CloseFiles(FilesOpen);
      FilesOpen = false;
      BuildScenario(ScenFilename);

      // form and execute the command to compile the applet
      try
      {
         RTime = Runtime.getRuntime();
         FullCompileCmd = new String(CompileCmd + " " + INSTALL_PATH
                                     + ScenFilename + ".java");

         if (CompileCmdSuffix != null)
            FullCompileCmd = FullCompileCmd + " " + CompileCmdSuffix;

         Proc = RTime.exec(FullCompileCmd);
      }
      catch(IOException e)
```

```
        {
          msg = new MsgBox(this, "ERROR", "Error " + e, " ");
          msg.show();
        }

        // create the file containing all files belonging to the message
        boolean foundString = FileList.find(ScenFilename + ".html");
        if (foundString == false)
          FileList.AddItem(ScenFilename + ".html");
        foundString = FileList.find(ScenFilename + ".class");
        if (foundString == false)
          FileList.AddItem(ScenFilename + ".class");

        BuildListFile(ScenFilename, FileList);
        return ScenFilename;
      }
      else
        return null;
}

public void BuildListFile(String filename, StrList List)
//*************************************************************************
// Create the file containing a list of all files used in the
// message.
//
// Parameters:
//    filename: name to save compressed message file as
//              (extension will be ".lst")
//   List: list of files used message
//*************************************************************************
{
  FileOutputStream FListStream = null;     // file list file
  PrintStream FListFile = null;            // file list file

  try
  {
    FListStream = new FileOutputStream(INSTALL_PATH + filename + "."
                                       + LIST_FILE_EXT);
    FListFile = new PrintStream(FListStream);
  }
  catch(IOException e)
  {
    msg = new MsgBox(this, "ERROR", "Error - Can Not Open File: ", filename
                     + ".txt");
    msg.show();
```

```
    }

    for (int i = 0; i < List.GetNumItems(); i++)
      FListFile.println(List.GetItem(i));

    FListFile.close();

    if (FListFile.checkError())
    {
      msg = new MsgBox(this, "ERROR", "ERROR - Can Not Close File: " + filename
                              + "." + LIST_FILE_EXT, " ");
      msg.show();
    }
}

public void DeleteMessage(boolean DeleteAll)
//*****************************************************************
// Gets name of a message to delete from the user and deletes all
// of the files associated with a message
//*****************************************************************
{
    DataInputStream FListFile = null;   // file list file
    boolean EndFile = false;            // true if at end of file
    String CurFName = null;             // current filename from list
    FileDialog GetFilenameBox;          // gets name of message to delete
    File DelFile;                       // a file to be deleted

    // get the filename of the message to present.
    GetFilenameBox = new FileDialog(this, "Select Scenario to Delete",
                                    FileDialog.LOAD);
    GetFilenameBox.setDirectory(".");

    if (ScenFilename != null)
      GetFilenameBox.setFile(ScenFilename);

    GetFilenameBox.show();
    ScenFilename = GetFilenameBox.getFile();
    String ScenFileDir = GetFilenameBox.getDirectory();

    if ((GetFilenameBox != null) && (ScenFilename != null))
    {
      ScenFilename = ScenFilename.substring(0, GetNameLen(ScenFilename));

      if (DeleteAll == true)
      {
```

```
    try
    {
       FListFile = new DataInputStream(new FileInputStream(INSTALL_PATH +
                                        ScenFilename + "." + LIST_FILE_EXT));
    }
    catch(IOException e)
    {
       msg = new MsgBox(this, "ERROR", "ERROR - file '" + ScenFilename
                              + "." + LIST_FILE_EXT + "' Not Found", " ");
       msg.show();
    }

    //* while not at end of TextFile, add the current character from the
    //* file to the text window and move to next char in file
    while (EndFile == false)
    {
       try
       {
          CurFName = FListFile.readLine();
          if (CurFName != null)
          {
             DelFile = new File(CurFName);
             DelFile.delete();
          }
          else
             EndFile = true;
       }
       catch(IOException e)
       {
          msg = new MsgBox(this, "ERROR", "ERROR " + e, " ");
          msg.show();
       }
    }

    try
    {   FListFile.close();      }
    catch(IOException e)
    {
       msg = new MsgBox(this, "ERROR", "ERROR - Can Not Close File: "
                              + ScenFilename  + ".txt", " ");
       msg.show();
    }
}
else
{
```

```
            DelFile = new File(ScenFilename + ".class");
            DelFile.delete();

            DelFile = new File(ScenFilename + ".html");
            DelFile.delete();
        }

        // delete the file containing filename list
        DelFile = new File(ScenFilename + "." + LIST_FILE_EXT);
        DelFile.delete();

        // delete the java source file if it exists
        if (DeleteSource == true)
        {
            DelFile = new File(ScenFilename + ".java");
            DelFile.delete();
            ScenFilename = null;
        }
    }
}

public void BuildWebPage(String ScenFilename)
//*************************************************************************
// Generate the html web page that embeds the applet named ScenFilename.
//*************************************************************************
{
    FileOutputStream htmlStream;     // html file
    PrintStream htmlFile = null;       // html file

    try
    {
        htmlStream = new FileOutputStream(INSTALL_PATH+ ScenFilename+".html");
        htmlFile = new PrintStream(htmlStream);
    }
    catch(IOException e)
    {
        msg = new MsgBox(this, "ERROR", "Error - can not open file: ", ScenFilename);
        msg.show();
    }


    htmlFile.println("<title>Slide Demonstration Mail Scenario</title>");
    htmlFile.println("<hr>");
    htmlFile.println("<applet code=\"" + ScenFilename + "\" width=" + AppletWidth
                     + " height=" + AppletHeight + ">");
```

```
   htmlFile.println("</applet>");
   htmlFile.println("<hr>");
   htmlFile.close();
}

public void ConcatFile(String FileToAppendName, PrintStream AppendFile)
//*************************************************************************
// Concatenate the contents of two files together.
//
// Parameters:
//       FileToAppendName: Name of file to append to the end of the
//                  the file AppendFile.
//       AppendFile: File to append the contents of AppendFile to.
//*************************************************************************
{
   String str = null;          // string read from file
   boolean EndFile = false;    // true if at end of file
   DataInputStream FileToAppend = null;  // stream for file to append

   try
   {
      FileToAppend = new DataInputStream(new FileInputStream(INSTALL_PATH
                                          + FileToAppendName));
   }
   catch(IOException e)
   {
      msg = new MsgBox(this, "ERROR", "ERROR - file '" + FileToAppendName
                              + "' Not Found", " ");
      msg.show();
   }

   //* while not at end of TextFile, add the current character from the
   //* file to the text window and move to next char in file
   while (EndFile == false)
   {
      try
      {
         str = FileToAppend.readLine();
         if (str != null)
         {
            AppendFile.println(str);
         }
         else
            EndFile = true;
      }
```

```
      catch(IOException e)
      {
         msg = new MsgBox(this, "ERROR", "ERROR " + e, " ");
         msg.show();
      }
   }

   try
   {
      FileToAppend.close();
   }
   catch(IOException e)
   {
      msg = new MsgBox(this, "ERROR", "ERROR - Can Not Close File: "
                              + FileToAppendName, " ");
      msg.show();
   }
}

public void BuildScenario(String ScenFilename)
//*************************************************************************
// Bring together the different pieces of applet code to create
// the entire applet for the scenario.
//
// Parameters:
//    ScenFilename: name of applet that implements scenario
//*************************************************************************
{
   FileOutputStream ScenFileStream = null;
   PrintStream ScenFile = null;

   try
   {
      ScenFileStream = new FileOutputStream(INSTALL_PATH + ScenFilename
                                                + ".java");
      ScenFile = new PrintStream(ScenFileStream);
   }
   catch(IOException e)
   {
      msg = new MsgBox(this, "ERROR", "Error - can not open file: ", ScenFilename);
      msg.show();
   }

   ConcatFile("ScenImports.txt", ScenFile);
   // write out the class definition line:
```

```
    ScenFile.println("public class " + ScenFilename +
                     " extends Applet implements Runnable");
    if (ScenFile.checkError())
    {
       msg = new MsgBox(this, "ERROR", "Scenario Generation Error", " ");
       msg.show();
    }

    ConcatFile("ScenStatDeclare.txt", ScenFile);
    ConcatFile("ScenDeclare.txt", ScenFile);
    ConcatFile("ScenInit.txt", ScenFile);
    ConcatFile("ScenRunEnd.txt", ScenFile);
    ConcatFile("ScenPaintBegin.txt", ScenFile);
    ConcatFile("ScenPaintEnd.txt", ScenFile);
    ConcatFile("ScenEnd.txt", ScenFile);

    ScenFile.close();
    if (ScenFile.checkError())
    {
       msg = new MsgBox(this, "ERROR", "ERROR - Can Not Close File: "
                + ScenFilename, " ");
       msg.show();
    }
}


public void AddBeginStateCode()
//*****************************************************************
// Add the code to create a new state in the scenario that implements
// a new slide or new non-slide part of message.
//*****************************************************************
{
   if (state > 0)   // 0 case code in files ScenInit.txt & ScenPaintBegin.txt
   {
      ScenRunFile.println("         else if(state == " + state + ")");
      ScenRunFile.println("         {");
      ScenRunFile.println("              getAppletContext().showStatus(\" \");");
      ScenRunFile.println("            ImageState = 0;");
      ScenRunFile.println("            repaint();");
      ScenPaintFile.println("       else if(state == " + state + ")");
      ScenPaintFile.println("       {");
      ScenPaintFile.println("          if (ImageState == 0)");
      ScenPaintFile.println("          {");
      ImageStateBegan = true;
   }
```

```
}

public void AddEndSlideCode()
//*********************************************************************
// Add the code that implements the end of a slide.
//*********************************************************************
{
   if (state > 0)
      ScenRunFile.println("          Previous.show();");

   ScenRunFile.println("          Next.show();");
   ScenRunFile.println("          ScenarioThread.suspend();");
   ScenRunFile.println("          TextBox.hide();");

   // since the next portion of the message may not belong to
   // a slide, then hide the slide movement buttons
   if (state > 0)
      ScenRunFile.println("          Previous.hide();");

   ScenRunFile.println("          Next.hide();");
   AddEndStateCode(SLIDE);
}

public void AddEndStateCode(int PartCreating)
//*********************************************************************
// Add the code to end a state associated with a slide or non-slide
// part of a message.
//
// Parameters:
//   PartCreating: type of complex message part being created
//            (SLIDE or NON-SLIDE)
//*********************************************************************
{
   if (PartCreating == NON_SLIDE)
   {
      ScenRunFile.println("          TextBox.hide();");
      ScenRunFile.println("          state = state + 1;");
   }

   ScenRunFile.println("          }");

   if (ImageStateBegan == true)
      ScenPaintFile.println("          }");    // End Image State
```

```
    ScenPaintFile.println("      }");      // End State
    state = state + 1;
    ImageState = 0;
    ImageStateBegan = false;
  }

  public void AddIncStateCode()
  //*************************************************************************
  // Move to the next state in the message.
  //*************************************************************************
  {
    ScenRunFile.println("          state += 1;");
    ScenPaintFile.println("          ImageState = 0;");
  }

  public void AddImageCode(String filename, String x, String y, String
                           width, String height)
  //*************************************************************************
  // Add the code to display an image in the applet.
  //
  // Parameter:
  //    filename: name of file containing image
  //    x:        x coordinate of upper left corner of image.
  //    y:        y coordinate of upper left corner of image.
  //    width:    width image will be displayed at (0 for normal)
  //    height:   height image will be displayed at (0 for normal)

//*************************************************************************
  {
    String varName = new String("image" + ImageNum);

    ScenDeclareFile.println("    private Image " + varName + ";");

    if (ImageStateBegan == false)
    {
      ScenRunFile.println("            ImageState++;");
      ScenRunFile.println("            repaint();");
      ImageState++;
      ScenPaintFile.println("          else if (ImageState == " + ImageState + ")");
      ScenPaintFile.println("          {");
      ImageStateBegan = true;
    }


    ScenPaintFile.println("            " + varName +
```

```
                                " = getImage(getCodeBase(), \"" + filename +  "\");");
    ScenPaintFile.println("            tracker.addImage(" + varName +  ", "
                          + ImageNum + ");");
    ScenPaintFile.println("            try");
    ScenPaintFile.println("            {");
    ScenPaintFile.println("               tracker.waitForID(" + ImageNum + ");");
    ScenPaintFile.println("            }");
    ScenPaintFile.println("            catch(InterruptedException e)");
    ScenPaintFile.println("            {");
    ScenPaintFile.println("            }");
    ScenPaintFile.print("          RetVal = g.drawImage(" + varName
                        + ", " + x + ", " + y);

    // if a width and height were given, then generate code to display
    // image at the desired width and height rather than the normal
    // width and height for the image.
    if ((height.length() != 0) && (width.length() != 0))
       ScenPaintFile.print(", " + width + ", " + height);

    ScenPaintFile.println(", this);");
    ImageNum++;

    boolean foundString = FileList.find(filename);
    if (foundString == false)
       FileList.AddItem(filename);
}

public void AddAudioCode(String filename)
//*************************************************************************
// Add the code to play an audio clip specified by filename in the applet.
//
// Parameter:
//    filename: name of file containing audio clip
//*************************************************************************
{
    String varName = new String("audio" + AudioNum);

    ScenDeclareFile.println("    private AudioClip " + varName + ";");

    ScenRunFile.println("          if (AudioCapable != false)");
    ScenRunFile.println("          {");
    ScenRunFile.println("             " + varName +
                        " = getAudioClip(getCodeBase(), \"" + filename + "\");");
    ScenRunFile.println("             " + varName + ".play();");
    ScenRunFile.println("             getAppletContext().showStatus(\"Audio: "
```

```
                        + filename + "; \");");
    ScenRunFile.println("            }");
    AudioNum++;

    boolean foundString = FileList.find(filename);
    if (foundString == false)
      FileList.AddItem(filename);
}

void AddSubTextCode()
{   ScenRunFile.println("        if (AudioCapable == false)");   }

public void AddTextCode(String filename, String x, String y,
                        String width, String height)
//*********************************************************************
// Add the code to display a text segmentin the applet.
//
// Parameter:
//    filename: name of file containing text
//    x:       x coordinate of upper left corner of text box.
//    y:       y coordinate of upper left corner of text box.
//    width:   width text box will be displayed at (0 for default)
//    height:  height text box will be displayed at (0 for default)
//*********************************************************************
{
    ScenRunFile.println("            ShowText(TextBox, \"" + filename +
                        "\", " + x + ", " + y + ", " + width + ", " + height + ");");

    boolean foundString = FileList.find(filename);
    if (foundString == false)
      FileList.AddItem(filename);
}

public void AddPointerCode(int x, int y, String PointerColor)
//*********************************************************************
// Add the code to display a pointer in the applet.
//
// Parameter:
//    x:       x coordinate that pointer will point to.
//    y:       y coordinate that pointer will point to.
//    PointerColor: color of the pointer.
//*********************************************************************
{
    ScenPaintFile.println("        ShowPointer(g, " + x + ", " + y + ", " +
                "Color." + PointerColor + ");");
```

```
}

public void AddWaitCode(String waitTime)
//*********************************************************************
// Add the code to implement a wait time (pause) in the playback of the
// applet.
// Parameter:
//    waitTime:  time to pause scenario
//*********************************************************************
{
   if (ImageStateBegan == true)
      ScenPaintFile.println("          }");

   ImageStateBegan = false;
   ScenRunFile.println("          try {ScenarioThread.sleep("
                + waitTime + "000);}");
   ScenRunFile.println("          catch (InterruptedException e) {}");
}

public void AddNumStatesCode()
//*********************************************************************
// Add the code for the constants that specify the number of states
// in the applet and the placement coordinates for the Next and Prev
// buttons.
//*********************************************************************
{
   int val;          // X or Y Pos for placement of a button

   ScenDeclareFile.println("    static final int NumStates = " + state + ";");
   ScenDeclareFile.println("    static final int PScale = " + PointerSize + ";");
   val = (int) (7* AppletWidth)/10;
   ScenDeclareFile.println("    static final int PrevBtnXPos = " + val + ";");
   val = (int) (9 * AppletHeight)/10;
   ScenDeclareFile.println("    static final int PrevBtnYPos = " + val + ";");
   val = (int) (8 * AppletWidth)/10 ;
   ScenDeclareFile.println("    static final int NextBtnXPos = " + val + ";");
}

public int GetFNameLen(String PathName)
//*********************************************************************
// Get the length of the filename from the path name specified by
// PathName.
//
// Parameter:
//    PathName: full pathname of: directory + filename
```

```
//*********************************************************************
{
   int len = 0;
   int i;
   boolean FNameStored = false;

   if (PathName != null)
   {
      i = PathName.length() - 1;

      while ((FNameStored == false) && (i >= 0))
      {
         // if a slash is encountered, then the filename has been traversed
         if ((PathName.charAt(i) == '\\') || (PathName.charAt(i) == '/'))
            FNameStored = true;
         else
         {
            len++;       // one character belonging to the filename
            i--;       // move to the previous character in the pathname
         }
      }
   }
   return (len);
}

public int Atoi(String str)
//*********************************************************************
// Return the integer value of string str.
// Parameter:
//    str: string to convert to integer.
//*********************************************************************
{
  int mult = 1;
  int val = 0;
  char ch;

  if (str != null)
     for (int i = str.length() - 1; i >= 0; i--)
     {
        ch = str.charAt(i);
        val = val + ((ch - '0') * mult);
        mult = mult * 10;
     }

  return val;
```

```
}

public boolean IsInt(String str)
//*************************************************************************
// Return true if the string str represents an integer.
// Parameter:
//    str: string to determine whether integer or not.
//*************************************************************************
{
   boolean valid = true;

   for (int i = 0; ((i < str.length()) && (valid)); i++)
      if ((str.charAt(i) > '9') || (str.charAt(i) < '0'))
         valid = false;
   return valid;
}

private void CloseFiles(boolean FilesOpen)
//*************************************************************************
// Close the applet code files that contain code that is generated
// as the scenario is specified.
// Parameter:
//    FilesOpen: true if generated code files are open
//*************************************************************************
{
   if (FilesOpen == true)
   {
      ScenRunFile.close();
      ScenPaintFile.close();
      ScenDeclareFile.close();
   }
}

public static void main(String args[])
{
   String MessPathname = null;      // full pathname of applet to play
   String MessFilename = null;      // filename of applet to play
   String MessDir = null;           // Directory where applet to play was stored
   boolean MessReceived = true;     // True if invoked to present mesage
   String FullAppletViewCmd = null; // entire command to play applet
   Process Proc;                    // variable representing the running process
   Runtime RTime;                   // runtime environment of running process
   ZipFileDecompress FileToUnzip= null;

   Frame ScenMailFrame = new ScenMail();
```

```
        try
        {   MessPathname = args[0].trim();      }
        catch(Exception e)
        {   MessPathname = null;   }

        if (MessPathname != null)
        {
          MessFilename = MessPathname.substring(MessPathname.length() -
                    ((ScenMail)ScenMailFrame).GetFNameLen(MessPathname));
          MessDir = MessPathname.substring(0, (MessPathname.length() -
                          MessFilename.length()));

          FileToUnzip= new ZipFileDecompress("", MessPathname);
          FileToUnzip.Decompress();

          // Rename the received file to move it from the temporary directory,
          // decompress the file, and present the message
          try
          {
            RTime = Runtime.getRuntime();
            MessFilename = MessFilename.substring(0, MessFilename.length()
                              - (((ScenMail)ScenMailFrame).MessageExtLen + 1));
            FullAppletViewCmd = new
                        String(((ScenMail)ScenMailFrame).AppletViewCmd
                        + " " + ((ScenMail)ScenMailFrame).INSTALL_PATH
                        + MessFilename + ".html");

            if (((ScenMail)ScenMailFrame).AppletViewCmdSuffix != null)
              FullAppletViewCmd = FullAppletViewCmd + " " +
              ((ScenMail)ScenMailFrame).AppletViewCmdSuffix;

            Proc = RTime.exec(FullAppletViewCmd);
          }
          catch(IOException e)
          {
            MsgBox msg = new MsgBox(ScenMailFrame, "ERROR", "Error " + e, "");
            msg.show();
          }
        }

        ScenMailFrame.resize(((ScenMail)ScenMailFrame).ApplicationWidth,
                    ((ScenMail)ScenMailFrame).ApplicationHeight);
        ScenMailFrame.show();
    }
}
```

Compose Message Class

```
import java.awt.*;
import java.io.*;
import ScenMail;
import ComposeMessageLayout;

class ComposeMessage extends Dialog
//---------------------------------------------------------------------------
// Class Compose Message
//
// Description:
// Implements the Compose Message Window that drives the composition of a
// message.
//---------------------------------------------------------------------------
{
   ScenMail parent;                  // main window object
   ComposeMessageLayout MyLayout;    // layout for controls

   public ComposeMessage(ScenMail TempParent)
   {
      super(TempParent, "Compose Message", false);
      setTitle("Compose Message");
      parent = TempParent;
      MyLayout = new ComposeMessageLayout(this);
      MyLayout.CreateControls();
   }

   public boolean action(Event evt, Object arg)
   {
      if(arg.equals("Done"))
         dispose();
      else if(arg.equals("Cancel"))
         dispose();
      else if(arg.equals("Add Media"))
      {
      // show the Add Media window
         if (parent.GetPartCreating() != parent.NON_SLIDE)
         {
            parent.AddBeginStateCode();
            parent.SetPartCreating(parent.NON_SLIDE);
         }
```

```
            AddMedia AddMBox = new AddMedia(parent, null);
            AddMBox.show();
        }
        else if (arg.equals("Add Slide"))
        {
        // show the Add Slide window
            if (parent.GetPartCreating() == parent.NON_SLIDE)
                parent.AddEndStateCode(parent.NON_SLIDE);

            parent.SetPartCreating(parent.SLIDE);
            ComposeSlide ComposeSlideBox = new ComposeSlide(parent);
            ComposeSlideBox.show();
        }
        else if (arg.equals("Add Wait"))
        {
        // show the Add Wait window
            if (parent.GetPartCreating() != parent.NON_SLIDE)
            {
                parent.AddBeginStateCode();
                parent.SetPartCreating(parent.NON_SLIDE);
            }

             AddWait AddWaitBox = new AddWait(parent, null);
             AddWaitBox.show();
        }

        return true;
    }


  public boolean handleEvent(Event evt)
  {
   if (evt.id == Event.WINDOW_DESTROY && evt.target == this)
        dispose();

    return super.handleEvent(evt);
  }
}
```

Compose Slide Class

```
import java.awt.*;
import java.io.*;
```

```
import ComposeSlideLayout;

class ComposeSlide extends Dialog
//---------------------------------------------------------------------------
// Class ComposeSlide
//
// Description:
// Implements the Compose Slide window that gets the filename of the image
// that the user wishes to display along with display size and location values
// and calls ScenMail methods to generate the applet code to display the image
//---------------------------------------------------------------------------
{
   private ScenMail parent;
   private ComposeSlideLayout MyLayout;

   public ComposeSlide(ScenMail TempParent)
   {
     super(TempParent, "Compose Slide", false);
     parent = TempParent;
     MyLayout = new ComposeSlideLayout(this);
     MyLayout.CreateControls();
     parent.AddBeginStateCode();
   }

   public boolean action(Event evt, Object arg)
   {
      if(arg.equals("Done"))
      {
         parent.AddEndSlideCode();
         dispose();
      }
      if(arg.equals("Cancel"))
      {
         dispose();
         return true;
      }

      else if (arg.equals("Add Media"))
      {
          AddMedia AddMediaBox = new AddMedia(parent, this);
          AddMediaBox.show();
          return true;
      }
      else if (arg.equals("Add Wait"))
      {
```

```
            AddWait AddWaitBox = new AddWait(parent, this);
            AddWaitBox.show();
            return true;
         }

         return false;
      }

      public boolean handleEvent(Event evt)
      {
         if (evt.id == Event.WINDOW_DESTROY && evt.target == this)
            dispose();

         return super.handleEvent(evt);
      }

      public void DisplayPart(String part)
      //*************************************************************************
      // Display an entry for a media part to be displayed inside the
      // current scenario listed for the slide.
      //*************************************************************************
      {   MyLayout.SlideList.addItem(part);   }
}
```

AboutDialog Class

```
import java.awt.*;

class AboutDialog extends Dialog
//-------------------------------------------------------------------
// class AboutDialog
//
// Description:
// Dialog window displayed to present information about Scenario Mail.
// Invoked through selection of About option from ScenarioMail File menu.
//-------------------------------------------------------------------
{
   public AboutDialog(Frame parent)
   {
      super(parent, "About DialogTest", true);
      Panel p1 = new Panel();
      p1.add(new Label("Scenario Mail, Version 1.0"));
      p1.add(new Label("Masters Thesis"));
```

```
        p1.add(new Label("By Jeff Holland, Oklahoma State University"));
        add("Center", p1);

        Panel p2 = new Panel();
        p2.add(new Button("Ok"));
        add("South", p2);
        resize(220, 150);
    }

    public boolean action(Event evt, Object arg)
    {
        if(arg.equals("Ok"))
        {
            dispose();
            return true;
        }
        return false;
    }

    public boolean handleEvent(Event evt)
    {
        if (evt.id == Event.WINDOW_DESTROY && evt.target == this)
            System.exit(0);

        return super.handleEvent(evt);
    }
}
```

Add Audio Class

```
import java.awt.*;
import java.io.*;
import ScenMail;
import ComposeSlide;
import AddAudioLayout;

class AddAudio extends Dialog
//-------------------------------------------------------------------------
// Class AddAudio
//
// Description:
// Implements the Add Audio window that gets the filename of the audio
// segment that the user wishes to play and calls ScenMail methods
```

```
// to generate the applet code to play the audio.
//-------------------------------------------------------------------------
{
   ScenMail parent;                  // main window object
   AddAudioLayout MyLayout;          // layout of window controls
   String Filename = null;           // name of audio segment file
   ComposeSlide SlideBox;            // compose slide object
   MsgBox msg;                       // error / warning message box
   boolean AudioCodeWritten = false;  // true when Audio code has been generated

   public AddAudio(ScenMail TempParent, ComposeSlide TempSlideBox)
   {
      super(TempParent, "Add Audio", false);
      setTitle("Add Audio");
      parent = TempParent;
      SlideBox = TempSlideBox;
      MyLayout = new AddAudioLayout(this);       // lay out the controls on the form
      MyLayout.CreateControls();
   }

   public boolean action(Event evt, Object arg)
   {
      if(arg.equals("OK"))
      {
      // Generate the code to play the selected audio segment
         Filename = MyLayout.FilenameTBox.getText().trim();

         if (Filename.length() > 0)
         {
            if (AudioCodeWritten == false)
            {
               parent.AddAudioCode(Filename);

               if (SlideBox != null)
                  SlideBox.DisplayPart("Audio (" + Filename + ")");
            }

            dispose();
         }
         else
         {
            msg = new MsgBox(parent, "Incomplete Input",
                                    " Please Enter Filename for Audio. ", " ");
            msg.show();
         }
```

```
    }
    else if(arg.equals("Cancel"))
      dispose();
    else if(arg.equals("Substitute Text"))
    {
      // Generate the code to play the selected audio segment, as well
      // as the code to play the substitute text when audio is not available
      Filename = MyLayout.FilenameTBox.getText().trim();

      if (Filename.length() > 0)
      {
        parent.AddAudioCode(Filename);

        if (SlideBox != null)
          SlideBox.DisplayPart("If Not Audio (" + Filename + ") Then");

        AddText AddTextBox = new AddText(parent, SlideBox, true);
        AddTextBox.show();
        dispose();
      }
      else
      {
        msg = new MsgBox(parent, "Incomplete Input",
                  "Please Specify Filename for Audio First.", " ");
        msg.show();
      }
    }
    else if(arg.equals("Browse"))
    {
      // display file dialog box and get name of audio file
      FileDialog GetFilenameBox = new FileDialog(parent, "Select Filename",
                                  FileDialog.LOAD);
      GetFilenameBox.setDirectory(".");
      GetFilenameBox.show();

      Filename = GetFilenameBox.getFile();
      MyLayout.FilenameTBox.setText(Filename);
    }

    return true;
}

public boolean handleEvent(Event evt)
{
    if (evt.id == Event.WINDOW_DESTROY && evt.target == this)
```

```
            dispose();
        return super.handleEvent(evt);
    }
}
```

AddImage Class

```
import java.awt.*;
import java.io.*;
import ScenMail;
import ComposeSlide;
import AddImageLayout;
import AddPointer;

class AddImage extends Dialog
//---------------------------------------------------------------------------
// Class AddImage
//
// Description:
// Implements the Add Image window that gets the filename of the image
// that the user wishes to display along with display size and location values
// and calls ScenMail methods to generate the applet code to display the image
//---------------------------------------------------------------------------
{
   ScenMail parent = null;          // main window object
   ComposeSlide SlideBox = null;     // compose slide object
   AddImageLayout MyLayout = null;   // layout for window controls
   String Filename = null;          // filename of image to show
   String X = null;                 // X coordinate for upper left corner of image
   String Y = null;                 // Y coordinate for upper left corner of image
   String Width = null;             // Width of image
   String Height = null;            // Height of image
   boolean ImageCodeWritten = false; // true if code generated to show image
   boolean ControlsValid = false;    // true if valid data for all text boxes
   MsgBox msg = null;               // error / warning message box

   public AddImage(ScenMail TempParent, ComposeSlide TempSlideBox)
   {
      super(TempParent, "Add Image", true);
      parent = TempParent;
      SlideBox = TempSlideBox;
      MyLayout = new AddImageLayout(this);
      MyLayout.CreateControls();
```

```
    // Add the pointer colors to the drop down scrolling box:
    MyLayout.ColorList.addItem("white");
    MyLayout.ColorList.addItem("yellow");
    MyLayout.ColorList.addItem("black");
    MyLayout.ColorList.addItem("red");
    MyLayout.ColorList.addItem("blue");
    MyLayout.ColorList.addItem("pink");
    MyLayout.ColorList.addItem("green");
    MyLayout.ColorList.addItem("cyan");
    MyLayout.ColorList.addItem("orange");
    MyLayout.ColorList.addItem("magenta");
    MyLayout.ColorList.addItem("gray");
    MyLayout.ColorList.addItem("lightGray");
    MyLayout.ColorList.addItem("darkGray");
    MyLayout.ColorList.select(0);
}

public void Initialize()
{
    MyLayout.FilenameTBox.setText("");
    MyLayout.XTBox.setText("");
    MyLayout.YTBox.setText("");
    MyLayout.WidthTBox.setText("");
    MyLayout.HeightTBox.setText("");
    ImageCodeWritten = false;
    ControlsValid = false;
}

public boolean action(Event evt, Object arg)
{
    if(arg.equals("Done"))
    {
        if (ImageCodeWritten == false)
        {
            TrimControls();
            DefaultCoord();

            // save control values to avoid effects of unwanted
            // changes by the user
            Filename = MyLayout.FilenameTBox.getText();
            X = MyLayout.XTBox.getText();
            Y = MyLayout.YTBox.getText();
            Width = MyLayout.WidthTBox.getText();
            Height = MyLayout.HeightTBox.getText();
            ControlsValid = ValidateControls();
```

```
        if (ControlsValid)
        {
          // generate the code to display the image
          ImageCodeWritten = WriteImageCode(parent, SlideBox, Filename,
                                X, Y, Width, Height);
          dispose();
        }
      }
      else
      dispose();
}
else if(arg.equals("Cancel"))
{
   dispose();
}
else if(arg.equals("Add Pointer"))
{
   if (ImageCodeWritten == false)
   {
      TrimControls();
      DefaultCoord();

      // save control values to avoid effects of unwanted
      // changes by the user
      Filename = MyLayout.FilenameTBox.getText();
      X = MyLayout.XTBox.getText();
      Y = MyLayout.YTBox.getText();
      Width = MyLayout.WidthTBox.getText();
      Height = MyLayout.HeightTBox.getText();

      ControlsValid = ValidateControls();

      if (ControlsValid)
      {
        // generate the code to display the image
        ImageCodeWritten = WriteImageCode(parent, SlideBox, Filename,
                                X, Y, Width, Height);
      }
   }

   if (ControlsValid)
   {
      AddPointer AddPointerBox = new AddPointer(parent, SlideBox,
                parent.Atoi(Width), parent.Atoi(Height), parent.Atoi(X),
                 parent.Atoi(Y), Filename, MyLayout.ColorList.getSelectedItem());
```

```
            AddPointerBox.show();
        }
    }
    else if(arg.equals("Browse"))
    {
        FileDialog GetFilenameBox = new FileDialog(parent, "Select Filename",
                                                   FileDialog.LOAD);
        GetFilenameBox.setDirectory(".");
        GetFilenameBox.show();
        Filename = GetFilenameBox.getFile();
        MyLayout.FilenameTBox.setText(Filename);
    }

    return true;
}

public boolean handleEvent(Event evt)
{
    if (evt.id == Event.WINDOW_DESTROY && evt.target == this)
        dispose();
    return super.handleEvent(evt);
}

private void DefaultCoord()
//*************************************************************************
// Set the default coordinates for image display
//*************************************************************************
{
    if (MyLayout.XTBox.getText().length() == 0)
        MyLayout.XTBox.setText("0");

    if (MyLayout.YTBox.getText().length() == 0)
        MyLayout.YTBox.setText("0");
}

private void TrimControls()
//*************************************************************************
//* Remove space before and after all text box values
//*************************************************************************
{
    MyLayout.FilenameTBox.setText(MyLayout.FilenameTBox.getText().trim());
    MyLayout.XTBox.setText(MyLayout.XTBox.getText().trim());
    MyLayout.YTBox.setText(MyLayout.YTBox.getText().trim());
    MyLayout.WidthTBox.setText(MyLayout.WidthTBox.getText().trim());
    MyLayout.HeightTBox.setText(MyLayout.HeightTBox.getText().trim());
```

```
}

private boolean ValidateControls()
//***************************************************************************
//* Return true if all controls on window have valid data.
//* Return false otherwise.
//***************************************************************************
{
  boolean valid = true;     // false if a text box value is invalid

  if (Filename.length() == 0)
  {
    msg = new MsgBox(parent, "Incomplete Input",
                                  " Please Enter Filename for Image. ", " ");
    msg.show();
    valid = false;
  }
  else if (!parent.IsInt(MyLayout.XTBox.getText()))
  {
    msg = new MsgBox(parent, "Error In Input", " X Is Invalid. ",
                          "Please Reenter.");
    msg.show();
    valid = false;
  }
  else if (!parent.IsInt(MyLayout.YTBox.getText()))
  {
    msg = new MsgBox(parent, "Error In Input", " Y Is Invalid. ",
                          "Please Reenter.");
    msg.show();
    valid = false;
  }
  else if (MyLayout.WidthTBox.getText().length() != 0)
    if (!parent.IsInt(MyLayout.WidthTBox.getText()))
    {
      msg = new MsgBox(parent, "Error In Input", "Width Is Invalid.",
                          "Please Reenter.");
      msg.show();
      valid = false;
    }
  else if (MyLayout.HeightTBox.getText().length() != 0)
    if (!parent.IsInt(MyLayout.HeightTBox.getText()))
    {
      msg = new MsgBox(parent, "Error In Input", "Height Is Invalid.",
                          "Please Reenter.");
      msg.show();
```

```
            valid = false;
        }
    return valid;
}

public boolean WriteImageCode(ScenMail parent, ComposeSlide SlideBox,
                    String Filename, String X, String Y,
                    String Width, String Height)
//*************************************************************************
// Display an entry in the Compose Slide window for the image and
// generate the applet code to display the image.
// Parameter:
//    parent:   main window object
//    SlideBox: Compose Slide windo object
//    Filename: name of file containing image
//    X:        x coordinate of upper left corner of image
//    Y:        y coordinate of upper left corner of image
//    Width:    width image will be displayed at (0 for normal)
//    Height:   height image will be displayed at (0 for normal)
//*************************************************************************
{
    if (SlideBox != null)
    {
        // for entry and display in the box on the Compose Slide Window
        String PartStr = "Image (" + Filename + ")" +
        " at (" + X + ", " + Y + ")";

        if ((Width.length() > 0) && (Height.length() > 0))
            PartStr = PartStr + "; " + Width + " X " + Height;

        SlideBox.DisplayPart(PartStr);
    }

    parent.AddImageCode(Filename, X, Y, Width, Height);
    return true;
}
}
```

AddMedia Class

```
import java.awt.*;
import java.io.*;
import ScenMail;
```

```
import ComposeSlide;

class AddMedia extends Dialog
//---------------------------------------------------------------------------
// Class AddMedia
//
// Description:
// Implements the Add Media window where the user specifies the type of
// media that they wish to present.  For the chosen media, this class
// invokes the appropriate collaborator to handle the request
//---------------------------------------------------------------------------
{
   private ScenMail parent;
   private ComposeSlide SlideBox;
   AddMediaLayout MyLayout;

   public AddMedia(ScenMail TempParent, ComposeSlide TempSlideBox)
   {
      super(TempParent, "Add Media", false);
      parent = TempParent;
      SlideBox = TempSlideBox;
      MyLayout = new AddMediaLayout(this);
      MyLayout.CreateControls();
   }

  public boolean action(Event evt, Object arg)
  {
   if(arg.equals("Cancel"))
   {
     dispose();
   }
   else if (arg.equals("Image"))
   {
     AddImage AddImageBox = new AddImage(parent, SlideBox);
     AddImageBox.show();
     dispose();
   }
   else if (arg.equals("Audio"))
   {
     AddAudio AddAudioBox = new AddAudio(parent, SlideBox);
     AddAudioBox.show();
     dispose();
   }
    else if (arg.equals("Text"))
   {
```

```
        AddText AddTextBox = new AddText(parent, SlideBox, false);
        AddTextBox.show();
        dispose();
      }

     return true;
    }

    public boolean handleEvent(Event evt)
    {
      if (evt.id == Event.WINDOW_DESTROY && evt.target == this)
        dispose();
      return super.handleEvent(evt);
    }
}
```

AddPointer Class

```
import java.awt.*;
import java.awt.image.*;
import java.net.*;
import ComposeSlide;
import ScenMail;

public class AddPointer extends Dialog
//---------------------------------------------------------------------------
// Class AddPointer
//
// Description:
// Displays the image that a user wishes to place a pointer on.
// Gets the location that the pointer will be displayed by getting the
// location where the user mouse clicks.
// Invokes ScenMail methods to generate applet code to display the pointer.
//---------------------------------------------------------------------------
{
    private Image image;          // image adding pointer to
    private String ImageFile = null;// filename of image adding pointer to
    private int ImageWidth;       // width to display image at
    private int ImageHeight;      // height to dislay image at
    private int XOffset;          // offset used to give X coordinate on applet
    private int YOffset;          // offset used to give Y coordinate on applet
    private ScenMail parent;      // main window object
    private ComposeSlide SlideBox; // compose slide box
```

```
private String PtrColor = null; // color of pointer
private MediaTracker MTracker;  // utility to manipulate displayal of the image
private MsgBox msg;             // error / warning message box

public AddPointer(ScenMail TempParent, ComposeSlide TempSlideBox,
            int TempImageWidth, int TempImageHeight,
            int TempXOffset, int TempYOffset,    String TempImageFile,
            String TempPtrColor)
{
   super(TempParent, "Click to Add Pointer", false);

   // store object for main window and for Compose Slide windo
   parent = TempParent;
   SlideBox = TempSlideBox;

   // store size and position that image will be displayed on applet
   ImageHeight = TempImageHeight;
   ImageWidth = TempImageWidth;
   XOffset = TempXOffset;
   YOffset = TempYOffset;

   ImageFile = TempImageFile;   // store image filename
   PtrColor = TempPtrColor;     // store pointer color

   // prepare to display image
   MTracker = new MediaTracker(this);
   image = Toolkit.getDefaultToolkit().getImage(ImageFile);
   MTracker.addImage(image, 0);
   try
   {
      MTracker.waitForID(0);
   }
   catch(InterruptedException e)
   {
      msg = new MsgBox(parent, "Image Exception", "Error: " + e, "");
      msg.show();
   }

   // if specific width and height were given, then display image
   // at that size
   if ((ImageWidth > 0) && (ImageHeight > 0))
      resize(ImageWidth, ImageHeight);
   // otherwise, get the normal size for the image and display at that size
   else
   {
```

```
        ImageWidth = image.getWidth(this);
        ImageHeight = image.getHeight(this);
        resize(ImageWidth, ImageHeight);
      }
    }

    public boolean mouseDown(Event MEvent, int x, int y)
    {
      int XPos;
      int YPos;

      XPos = x + XOffset;
      YPos = y + YOffset;

      // Generate the pointer code and add entry to box on
      // compose slide window
      parent.AddPointerCode(XPos, YPos, PtrColor);

      if (SlideBox != null)
        SlideBox.DisplayPart("Pointer (" + PtrColor + ") at (" + XPos + ", " +
                    YPos + ")");
      dispose();
      return true;
    }

    public boolean handleEvent(Event evt)
    {
      if (evt.id == Event.WINDOW_DESTROY && evt.target == this)
        dispose();

      return super.handleEvent(evt);
    }

    public void update(Graphics g)
    {
      paint(g);
    }

    public void paint(Graphics g)
    {
      // display image
      g.drawImage(image, 0, 0, ImageWidth, ImageHeight, this);
    }
}
```

AddText Class

```
import java.awt.*;
import java.io.*;
import ScenMail;
import ComposeSlide;
import AddTextLayout;

class AddText extends Dialog
//---------------------------------------------------------------------------
// Class AddText
//
// Description:
// Implements the Add Text window that gets the filename of the text
// that the user wishes to display along with display size and location values
// and calls ScenMail methods to generate the applet code to display the text
//---------------------------------------------------------------------------
{
  ScenMail parent;              // main window object
  ComposeSlide SlideBox;        // Compose Slide window object
  AddTextLayout MyLayout;       // layout for window controls
  String Filename = null;       // name of file containing text to present
  String X = null;              // X coordinate of upper left corner of text box
  String Y = null;              // Y coordinate of upper left corner of text box
  String Width = null;          // width of text box to present
  String Height = null;         // height of text box to present
  boolean TextCodeWritten = false; // true if code generated to present the text
  boolean ControlsValid;        // true if valid data in all controls
  boolean Sub;                  // true if text will be substitute for audio
  MsgBox msg;                   // warning / error message box
  static String DefaultHeight = "200";  // default height for text box
  static String DefaultWidth = "300";   // default width for text box

  public AddText(ScenMail TempParent, ComposeSlide TempSlideBox,
                 boolean TempSub)
  {
    super(TempParent, "Add Text", false);
    parent = TempParent;
    SlideBox = TempSlideBox;
    Sub = TempSub;
    MyLayout = new AddTextLayout(this);
    MyLayout.CreateControls();
  }
```

```
public boolean action(Event evt, Object arg)
{
  if(arg.equals("OK"))
  {
    if (TextCodeWritten == false)
    {
      TrimControls();
      DefaultCoord();

      // save control values to avoid effects of unwanted changes
      Filename = MyLayout.FilenameTBox.getText();
      X = MyLayout.XTBox.getText();
      Y = MyLayout.YTBox.getText();
      Width = MyLayout.WidthTBox.getText();
      Height = MyLayout.HeightTBox.getText();
      ControlsValid = ValidateControls();

      if (ControlsValid)
      {
        if (Sub == true)
          parent.AddSubTextCode();

        TextCodeWritten = WriteTextCode(parent, SlideBox, Filename,
                                X, Y, Width, Height, Sub);
        dispose();
      }
    }
    return true;
  }
  else if(arg.equals("Cancel"))
  {
    dispose();
    return true;
  }
  else if(arg.equals("Browse"))
  {
    FileDialog GetFilenameBox = new FileDialog(parent, "Select Filename",
                                                FileDialog.LOAD);
    GetFilenameBox.setDirectory(".");
    GetFilenameBox.show();
    Filename = GetFilenameBox.getFile();
    MyLayout.FilenameTBox.setText(Filename);
    return true;
  }
```

```
    return true;
}

public boolean handleEvent(Event evt)
{
    if (evt.id == Event.WINDOW_DESTROY && evt.target == this)
        dispose();
    return super.handleEvent(evt);
}

private void DefaultCoord()
//*************************************************************************
// Set the default coordinates for image display
//*************************************************************************
{
    if (MyLayout.XTBox.getText().length() == 0)
        MyLayout.XTBox.setText("0");

    if (MyLayout.YTBox.getText().length() == 0)
        MyLayout.YTBox.setText("0");

    if (MyLayout.WidthTBox.getText().length() == 0)
        MyLayout.WidthTBox.setText(DefaultWidth);

    if (MyLayout.HeightTBox.getText().length() == 0)
        MyLayout.HeightTBox.setText(DefaultHeight);
}

private void TrimControls()
//*************************************************************************
//* Remove space before and after all text box values
//*************************************************************************
{
    MyLayout.FilenameTBox.setText(MyLayout.FilenameTBox.getText().trim());
    MyLayout.XTBox.setText(MyLayout.XTBox.getText().trim());
    MyLayout.YTBox.setText(MyLayout.YTBox.getText().trim());
    MyLayout.WidthTBox.setText(MyLayout.WidthTBox.getText().trim());
    MyLayout.HeightTBox.setText(MyLayout.HeightTBox.getText().trim());
}

private boolean ValidateControls()
//*************************************************************************
// Return true if valid data has been entered for all text box controls.
// Return false othewise.
//*************************************************************************
```

```
{
   boolean valid = true;

   if (Filename.length() == 0)
   {
      msg = new MsgBox(parent, "Incomplete Input",
                              "  Please Enter Filename for Image.  ", " ");
      msg.show();
      valid = false;
   }
   else if (!parent.IsInt(MyLayout.XTBox.getText()))
   {
      msg = new MsgBox(parent, "Error In Input", "X Is Invalid.", "Please Reenter.");
      msg.show();
      valid = false;
   }
   else if (!parent.IsInt(MyLayout.YTBox.getText()))
   {
      msg = new MsgBox(parent, "Error In Input", "Y Is Invalid.", "Please Reenter.");
      msg.show();
      valid = false;
   }
   else if (MyLayout.WidthTBox.getText().length() != 0)
      if (!parent.IsInt(MyLayout.WidthTBox.getText()))
      {
         msg = new MsgBox(parent, "Error In Input", "Width Is Invalid.",
                              "Please Reenter.");
         msg.show();
         valid = false;
      }

   else if (MyLayout.HeightTBox.getText().length() != 0)
      if (!parent.IsInt(MyLayout.HeightTBox.getText()))
      {
         msg = new MsgBox(parent, "Error In Input", "Height Is Invalid.",
                              "Please Reenter.");
         msg.show();
         valid = false;
      }
   return valid;
}

public boolean WriteTextCode(ScenMail parent, ComposeSlide SlideBox,
                     String Filename, String X, String Y,
                     String Width, String Height, boolean Sub)
```

```
    //*************************************************************************
    // Display an entry in the Compose Slide window for the text part and
    // generate the applet code to display the box presenting the text
    // Parameter:
    //    parent:   main window object
    //    SlideBox: Compose Slide window object
    //    Filename: name of file containing text box
    //    X:        x coordinate of upper left corner of text box
    //    Y:        y coordinate of upper left corner of text box
    //    Width:    width of text box (0 for default)
    //    Height:   height of text box (0 for default)
    //    Sub:       True if text will be used as substitution for audio
    //*************************************************************************
    {
      String PartStr = new String(""); // Displayed in the Compose Slide window box

      if (SlideBox != null)
      {
        // form the entry for the Compose Slide window box and display it
        if (Sub == true)
          PartStr = "    ";
        PartStr = PartStr + "Text (" + Filename + ")" + "; at (" + X + ", " + Y + ")";
        if ((Width.length() > 0) && (Height.length() > 0))
          PartStr = PartStr + "; " + Width + " X " + Height;

        SlideBox.DisplayPart(PartStr);
      }

      // generate the applet code
      parent.AddTextCode(Filename, X, Y, Width, Height);
      return true;
    }
}
```

AddWait Class

```
import java.awt.*;
import java.io.*;
import ScenMail;
import ComposeSlide;
import AddImageLayout;

class AddWait extends Dialog
```

```
//---------------------------------------------------------------------------
// Class AddWait
//
// Description:
// Implements the Add Wait window that gets the wait time to pause in a
// scenario and calls ScenMail methods to generate the applet code to pause.
//---------------------------------------------------------------------------
{
  ScenMail parent;              // main window object
  ComposeSlide SlideBox;        // Compose Slide window object
  AddWaitLayout MyLayout;       // layout of controls for window
  MsgBox msg;                   // warning / error message box

  public AddWait(ScenMail TempParent, ComposeSlide TempSlideBox)
  {
    super(TempParent, "Add Wait", false);
    parent = TempParent;
    SlideBox = TempSlideBox;
    MyLayout = new AddWaitLayout(this);
    MyLayout.CreateControls();
  }

  public boolean action(Event evt, Object arg)
  {
    if(arg.equals("OK"))
    {
      if (MyLayout.WaitTimeTBox.getText().trim().length() > 0)
      {
        parent.AddWaitCode(MyLayout.WaitTimeTBox.getText().trim());

        if (SlideBox != null)
          SlideBox.DisplayPart("Wait(" + MyLayout.WaitTimeTBox.getText().trim()
                                       + ")");

        dispose();
      }
      else
      {
        msg = new MsgBox(parent, "Incomplete Input", "  Please Enter Wait Time.  ",
                                 " ");
        msg.show();
      }
    }
    else if(arg.equals("Cancel"))
      dispose();
```

```
    return true;
  }

  public boolean handleEvent(Event evt)
  {
   if (evt.id == Event.WINDOW_DESTROY && evt.target == this)
       dispose();
    return super.handleEvent(evt);
  }
}
```

MsgBox Class

```
import java.awt.*;

class MsgBox extends Dialog
//-------------------------------------------------------------------------
// Class MsgBox
//
// Description:
// Message box used display messages such as warnings or error messages.
//-------------------------------------------------------------------------
{
  boolean exit = false;

  public MsgBox(Frame parent, String Title, String MsgLine1, String MsgLine2)
  // ************************************************************************
  // Initialize the message box to be displayed.
  //
  // Parameters:
  //     parent: main window
  //     Title:  title of the message box
  //     MsgLine1: first line of message    to display
  //     MsgLine2: second line of message to display
  //************************************************************************
  {
    super(parent, Title, true);
    Panel p1 = new Panel();
    p1.add(new Label(MsgLine1));
    p1.add(new Label(MsgLine2));
    add("Center", p1);
    Panel p2 = new Panel();
```

```
        p2.add(new Button("OK"));
        add("South", p2);

        if (Title.toUpperCase().compareTo("ERROR") == 0)
            exit = true;

        resize(220, 150);
    }

    public boolean action(Event evt, Object arg)
    {
        if(arg.equals("OK"))
        {
            if (exit)
                System.exit(0);
            else
                dispose();

            return true;
        }

        return true;
    }

    public boolean handleEvent(Event evt)
    {
        if (evt.id == Event.WINDOW_DESTROY && evt.target == this)
            dispose();
        return super.handleEvent(evt);
    }
}
```

StrList Class

```
import ScenMail;

class StrList
//---------------------------------------------------------------------------
// Class StrList
//
// Description:
// List of strings and the operations to perform the list, which are:
//      find, AddItem, GetItem, GetNumItems, PrintList
```

```
//-----------------------------------------------------------------------
{
   private String[] List;      // List of files used in scenario
   private int NumItems;       // Number of files used in scenario
   private int MaxItems;       // Maximum number of files in scenario
   private MsgBox msg;         // Form to display messages
   private ScenMail parent;    // ScenMail main window object

   public StrList(ScenMail TempParent, int NewMaxItems)
   {
      MaxItems = NewMaxItems;
      List = new String[MaxItems];
      NumItems = 0;
      parent = TempParent;
   }

   public boolean find(String Str)
   //*************************************************************************
   // Return true if string Str is found in list List.
   // Return false is string is not found in list.
   //
   // Parameters:
   //       Str: String to find in list
   //*************************************************************************
   {
      boolean Found = false;

      Str = Str.trim();

      for (int i = 0; i < NumItems; i++)
      {
         if (List[i].compareTo(Str) == 0)
            Found = true;
      }

      return Found;
   }

   public boolean AddItem(String Str)
   //*************************************************************************
   // Add the item Str to the list
   //
   // Parameters:
   //       Str: String to add to the list
   //*************************************************************************
```

```
    {
      if (NumItems < (MaxItems - 1))
        List[NumItems++] = Str.trim();
      else
      {
        msg = new MsgBox(parent, "ERROR", "Maximum File Limit Reached", " ");
        msg.show();
        return false;
      }

      return true;
    }

    public String GetItem(int Loc)
    //*************************************************************************
    // Return item at specified location (Loc) in list.
    //
    // Parameters:
    //       Loc: Location of item to return.
    //*************************************************************************
    {       return List[Loc];       }

    public int GetNumItems()
    //*************************************************************************
    // Return the number of items in the list.
    //*************************************************************************
    {       return NumItems;        }

    public void PrintList()
    //*************************************************************************
    // Output the contents of the list.
    //*************************************************************************
    {
      for (int i = 0; i < NumItems; i++)
        System.out.println("    Item: " + i + List[i]);
    }
}
```

ZipFileCompress Class

```
import java.util.zip.*;
import java.io.*;
import ScenMail;
```

```
class ZipFileCompress
//---------------------------------------------------------------------
// Class PointerColor
//
// Description:
// Compress a file or group of files into a single compressed file using
// the zip format used by PkZip.
//---------------------------------------------------------------------
{
   private ZipOutputStream ZipStream = null;
   private MsgBox msg;         // Message box for errors and warnings
   private ScenMail parent;    // ScenMail main window object
   static int WAIT_MAX = 100;  // maximum number of times to wait for compile

   public ZipFileCompress(ScenMail TempParent, String ZipFilename)
   //*********************************************************************
   // Create and initialize the empty zip file and prepare to write the
   // first entry (file) to the zip file.
   //
   // Parameters:
   //        ZipFilename: name of compressed (zip) file
   //*********************************************************************
   {
      FileOutputStream ZipFileStream = null;
      parent = TempParent;

      try
      {  ZipFileStream = new FileOutputStream(ZipFilename);     }
      catch(IOException e)
      {
         msg = new MsgBox(parent, "ERROR", "Zip File Stream", "Could Not Be
                                  Created");
         msg.show();
      }

      ZipStream = new ZipOutputStream(ZipFileStream);

      try
      {
         ZipStream.setMethod(ZipStream.DEFLATED);
         ZipStream.setLevel(0);
      }
      catch(IllegalArgumentException e)
      {
         msg = new MsgBox(parent, "ERROR", "Zip Level Error: ", " ");
```

```
      msg.show();
      System.out.println("Zip Error: " + e);
    }
}

public void AddFile(String AddFilename)
//*************************************************************************
// Compress and add the contents of a file to the zip file being created
//
// Parameters:
//        AddFilename: Name of file to compress and add.
//*************************************************************************
{
   int MAX_BUF = 100;    // maximum length of buffer to hold data
   String str = null;          // string read from file
   boolean EndFile = false;    // true if at end of file
   FileInputStream FileToAdd= null;  // stream for file to append
   byte Buf[];                 // buffer to hold data from file
   Buf = new byte[MAX_BUF];
   int BufLen = 0;             // number of bytes in buffer
   ZipEntry NextEntry = new ZipEntry(AddFilename);
   boolean Waiting = true;       // file has not been opened
   int WaitCount;              // # of tries to open
   boolean StillCompiling=false;  // if true, ".class" file still in use

   try
   {   ZipStream.putNextEntry(NextEntry);    }
   catch(IOException e)
   {
      msg = new MsgBox(parent, "ERROR", "Cannot Add Next", "Entry to Zip File");
      msg.show();
   }

   WaitCount = 0;
   // if compile is not finished, wait for class file
   while ((Waiting) && (WaitCount < WAIT_MAX))
   {
      StillCompiling = false;
      try
      {
         FileToAdd = new FileInputStream(AddFilename);
         System.out.println("Adding file: " + AddFilename);
      }
      catch(IOException e)
      {
```

```
      if (AddFilename.endsWith(".class"))
      {
         StillCompiling = true;
         WaitCount++;
         try {parent.ScenarioThread.sleep(1000);}
         catch (InterruptedException ex) {}
      }
      else
      {
         msg = new MsgBox(parent, "ERROR", "File '" + AddFilename +
                                  "' Not Found", " ");
         msg.show();
      }
   }

   if (StillCompiling == false)
      Waiting = false;
}

//* while not at end of file to add, read one buffer at time and write
//* the buffer to the compressed message file in compressed form
while (EndFile == false)
{
   try
   {
      BufLen = FileToAdd.read(Buf);
      if (BufLen != -1)
         ZipStream.write(Buf, 0, BufLen);
      else
         EndFile = true;
   }
   catch(IOException e)
   {
      msg = new MsgBox(parent, "ERROR", "ERROR " + e, " ");
      msg.show();
   }
}

try
{
   FileToAdd.close();
   ZipStream.closeEntry();
}
catch(Exception e)
{
```

```
            msg = new MsgBox(parent, "ERROR", "Can Not Close File: " +
                        AddFilename, " ");
            msg.show();
        }
    }

    public void Save()
    //************************************************************************
    // Close and save the compressed file
    //************************************************************************
    {
        try
        {
            ZipStream.finish();
            ZipStream.close();
        }
        catch(IOException e)
        {
            msg = new MsgBox(parent, "ERROR", "Zip File Close Error", " ");
            msg.show();
        }
    }
}
```

ZipFileDecompress Class

```
import java.util.zip.*;
import java.util.Enumeration;
import java.io.*;
import ScenMail;

class ZipFileDecompress
//-------------------------------------------------------------------------
// Class PointerColor
//
// Description:
// Extract the files stored in a zip format compressed file.
//-------------------------------------------------------------------------
{
    private ZipFile UnzipFile;
    private String BasePath = null;
    private String ZipFilename = null;  //
    private ZipInputStream ZipStream = null;    // Output stream to write zip file contents
```

```
public ZipFileDecompress(String TempBasePath, String TempZipFilename)
//*************************************************************************
// Decompress file named ZipFilename, extracting the files relative to
// BasePath
//
// Parameters:
//     ZipFilename: name of compressed (zip) file
//       BasePath:    path where files will be stored relative to
//*************************************************************************
{
   BasePath = TempBasePath;
   ZipFilename = TempZipFilename;

   try   {   UnzipFile = new ZipFile(ZipFilename);   }
   catch (IOException e)
   {
      System.out.println("ERROR - Zip File: " + ZipFilename +
                         " Could Not Be Opened");
   }
}

void Decompress()
{
   Enumeration ZipEntries;
   try
   {   ZipStream = new ZipInputStream(new FileInputStream(ZipFilename));   }
   catch(Exception e)
   {   System.out.println("Zip File Stream Could Not Be Created");      }

   ZipEntries = UnzipFile.entries();

   while (ZipEntries.hasMoreElements())
   {
      try
      {
         ZipEntries.nextElement();
         ExtractFile();
      }
      catch (Exception e)
      {
         System.out.println("Error - Element " + e + " Could Not Be Extracted");
      }
   }

   try
```

```
    {   ZipStream.close();      }
    catch(IOException e)
    {   System.out.println("Zip File Close Error");     }
}

void ExtractFile()
//*************************************************************************
// Extract the specified file from the compressed zip file
//
// Parameters:
//        ExtractFilename: Name of file to extract
//*************************************************************************
{
    int MAX_BUF = 100;                          // max. length of buffer to hold data
    String str = null;                          // string read from file
    boolean EndFile = false;                    // true if at end of file
    DataInputStream FileToExtract= null;        // stream for file to append
    byte Buf[];                                 // buffer to hold data from file
    Buf = new byte[MAX_BUF];
    int BufLen = 0;                             // number of bytes in buffer
    FileOutputStream ExtractedFile = null;      // file extracted from zip file
    ZipEntry UnzipEntry = null;

    try
    {
      // Get next entry (file) from zip file
      UnzipEntry = ZipStream.getNextEntry();
    }
    catch(Exception e)
    {
      System.out.println("Cannot Extract Next Entry from Zip File");
    }

    try
    {
      ExtractedFile = new FileOutputStream(BasePath + UnzipEntry.getName());
    }
    catch(IOException e)
    {
      System.out.println(UnzipEntry.getName() + " Could Not be Extracted");
    }

    //* while not at end of file to add, read one buffer at time and write
    //* the buffer to the compressed message file in compressed form
    while (EndFile == false)
```

```
        {
          try
          {
            BufLen = ZipStream.read(Buf);
            if (BufLen != -1)
              ExtractedFile.write(Buf, 0, BufLen);
            else
              EndFile = true;
          }
          catch(Exception e)
          {
            System.out.println("ERROR: " + e);
          }
        }

        try
        {
          ExtractedFile.close();
          ZipStream.closeEntry();
        }
        catch(IOException e)
        {
          System.out.println("ERROR - Can Not Close Extracted File: " + e);
        }
      }
    }
```

APPENDIX B

Applet Template Code

ScenImports.txt

```
import java.awt.*;
import java.applet.*;
import java.awt.image.*;
import java.net.*;
import java.io.*;
import java.net.MalformedURLException;
```

ScenStatDeclare.txt

```
{
   private boolean RetVal;                          // Value returned by method call
   private Button Previous;                         // Button used to move to previous slide
   private Button Next;                             // Button used to move to next slide
   private MediaTracker tracker;                    // Tracks the status of image objects
   private boolean AudioCapable;                    // False if no sound capability is available
   public Thread ScenarioThread = null;             // applet thread
   private int state = 0;                           // Indicates the part of the message
                                                    // currently being played
   private int ImageState = 0;                      // Indicates part of state consisting of
                                                    // the group of images prior to the next
                                                    // wait
   private TextArea TextBox;                        // Used to display a text part of a message
   static final int SCENARIO_DONE = -999;           // End of message sentinel
   static final int ButtonWidth = 70;               // Width of next and previous buttons
   static final int ButtonHeight = 35;              // Height of next and previous buttons

   //* Scenario specific declarations will follow
```

ScenInit.txt

```
public void init()
{
   URL TextData = null;                     // Description of textfile Audio.cntl
   InputStream AudioCntlStream = null;      // Audio.cntl stream to get audio flag from
   char AudioFlag = 'Y';                    // "N" for not audio capable

   // Start at first part of message being played
   state = 0;
   tracker = new MediaTracker(this);
```

```
    // No flow layout manager for applet frame
    setLayout(null);

    // Add next and previous buttons to the frame - Each will be visible
    // as appropriate
    Previous = new Button("Previous");
    add(Previous);
    Previous.reshape(PrevBtnXPos, PrevBtnYPos, ButtonWidth, ButtonHeight);
    Next = new Button("Next");
    add(Next);
    Next.reshape(NextBtnXPos, PrevBtnYPos, ButtonWidth, ButtonHeight);
    Previous.hide();
    Next.hide();

// determine if the presenting machine has sound capability
    AudioCapable = true;    // assume sound capable until proven false

    // prepare the text file for reading:
    try
    {
       TextData = new URL(getCodeBase(), "Audio.cntl");
    }
    catch (Exception e)
    {
       showStatus("Audio.cntl file error ==> Assuming Audio Capable");
    }

    if (TextData != null)
    {
       try
       {
          AudioCntlStream = TextData.openStream();
       }
       catch(IOException e)
       {
          showStatus("Audio.cntl file error ==> Assuming Audio Capable");
       }

       if (AudioCntlStream != null)
       {
          // get the flag value from the file
          try
          {
             AudioFlag = (char) AudioCntlStream.read();
          }
```

```
            catch(IOException e)
            {
               showStatus("Audio.cntl file error ==> Assuming Audio Capable");
            }

            // if file contains flag value of "N", then machine is not audio capable
            if (AudioFlag != -1)
            {
               if ((AudioFlag == 'n') || (AudioFlag == 'N'))
                  AudioCapable = false;
            }
         }
      }
   }

   public boolean action(Event evt, Object arg)
   {
    // if next button was clicked move to the next slide

    if (arg.equals("Done"))
     System.exit(0);

    else if (arg.equals("Next"))
    {
       if (state == NumStates)
         state = SCENARIO_DONE;
       else
         state += 1;

       if (state > 1)
          Previous.hide();

       Next.hide();

       // Continue with rest of scenario:
       ScenarioThread.resume();
    }

    // if next button was clicked move to the previous slide
    else if (arg.equals("Previous"))
    {
       if (state > 0)
         state -= 1;

       // Continue with rest of scenario:
```

```
    ScenarioThread.resume();
  }

  return true;
}

public void run()
{
  // Prepare text window used to display all text parts of message:
  TextBox = new TextArea(8, 40);
  add("Center", TextBox);

  // While the entire scenario has not been played, play the next part of the
  // message (scenario):
  while (state < NumStates)
  {
     if (state < (NumStates - 1))
        Next.setLabel("Next");
     else
        Next.setLabel("Done");

     if (state == 0)
     {
        ImageState = 0;
        repaint();          // Show images and pointers for this part
```

ScenPaintBegin.txt

```
  }
}

public void paint(Graphics g)
{
  //* The code for displaying the images and pointers for each state (phase)
  //* of the scenario will follow:
  if (state == 0)
  {
     if (ImageState == 0)
     {
```

ScenEnd.txt

```
}

public void start()
 {
   //* Create the main thread for the applet
   if (ScenarioThread == null)
   {
      ScenarioThread = new Thread(this);
      ScenarioThread.start();
   }
}

void ShowText(TextArea TextBox, String TextFile, int x, int y, int width, int height)
//*********************************************************************
// Display the contents of a text file whose name is given by
// parameter TextFile in a text window specified by parameter TextBox.
// The location to display the text window is given by parameters x
// and y.  The window will have height and width specified by
// parameters width and height.
//
// Parameters:
//       TextBox: Window to display text in
//       TextFile: File containing text to display
//       x: x location of upper left corner of TextBox.
//       y: y location of upper left corner of TextBox.
//       width: width of TextBox
//       height: width of TextBox
//*********************************************************************
{
   boolean EndFile;                 // true when at end of textfile
   int ch=' ';                      // current char in textfile
   String Str;                      // string to add to textbox
   URL TextData = null;             // Description of textfile
   InputStream TextStream = null;   // stream to get text from

   //* display text window:
   TextBox.reshape(x, y, width, height);
   TextBox.show();


   //* prepare the text file for reading:
   try
   {
```

```
        TextData = new URL(getCodeBase(), TextFile);
    }
    catch (MalformedURLException e)
    {
        System.out.println("Error " + e);
    }

    try
    {
        TextStream = TextData.openStream();
    }
    catch(IOException e)
    {
        showStatus("Cannot Display Contents of Text File: " + TextFile);
    }

    //* erase any text from a previous part of scenario
    TextBox.selectAll();
    TextBox.replaceText("", 0, TextBox.getSelectionEnd());

    EndFile = false;

    //* while not at end of TextFile, add the current character from the
    //* file to the text window and move to next char in file
    while (EndFile == false)
    {
        try
        {
            ch = TextStream.read();
        }
        catch(IOException e)
        {
            showStatus("Error " + e);
        }

      if (ch != -1)
      {
          Str = "" + (char)ch;
          TextBox.appendText(Str);
      }
      else
          EndFile = true;
    }
}
```

```
void ShowPointer(Graphics g, int x, int y, Color PColor)
//*********************************************************************
// Display a pointer of color PColor at location x, y.
//
// Parameters:
//      x: x location to display pointer
//      y: y location to display pointer
//      Pcolor: color to color interior of pointer
//*********************************************************************
{
   Polygon Pointer = new Polygon();   // polygon used for pointer
   int xPos, yPos;                    // coordinates of next point
                                      // to plot when displaying pointer

// Pointer will be composed of 7 points.  These 7 points are labeled 1 to 7
// as each point is added in counterclockwise order starting with the point
// at the "point" of the pointer.

//1 - Add point at "point" of pointer
     Pointer.addPoint(x, y);

//2 - Add next point
     xPos = x - (int)(0.07 * PScale);
     yPos = y + (int)(1.1 * PScale);
     Pointer.addPoint(xPos, yPos);

//3 - Add next point
     xPos = x - (int)(0.2 * PScale);
     yPos = y + (int)(0.8 * PScale);
     Pointer.addPoint(xPos, yPos);

//4 - Add next point
     xPos = x - (int)(0.6 * PScale);
     yPos = y + (int)(1.5 * PScale);
     Pointer.addPoint(xPos,  yPos);

//5 - Add next point
     xPos = x - (int)(0.9 * PScale);
     yPos = y + (int)(1.3 * PScale);
     Pointer.addPoint(xPos, yPos);

//6 - Add next point
     xPos = x - (int)(PScale/2);
     yPos = y + (int)(0.7 * PScale);
     Pointer.addPoint(xPos, yPos);
```

```
//7 - Add next point
        xPos = x - (int)(0.8 * PScale);
        yPos = y + (int)(0.8 * PScale);
        Pointer.addPoint(xPos, yPos);

        // draw black outline around pointer:
        g.setColor(Color.black);
        g.drawPolygon(Pointer);

        // fill interior with color PColor
        g.setColor(PColor);
        g.fillPolygon(Pointer);
    }
}
```

APPENDIX C

Example Scenario Code

Scenario

An example scenario that consists of four slides is shown in "pseudocode" below. Statements beginning with "//" are comments.

```
Begin Slide 1
   // show text in window at location (0, 300) with width 400 and height 100.
   TextBox(CircProb.txt) at (0, 300); 400 X 100)
   // pause for 3 seconds before proceeding
   Wait(3)
   // Show image in upper left corner with width 200 and height 200.
   Image(CircProb.jpeg) at (0, 0); (200 X 200)
   // Place a white pointer and a black pointer on the CircProb.jpeg
   // image at locations (40, 22) and (145, 115) respectively.
   Pointer(White) at (174, 76)
   Pointer(Black) at (24, 38)
End Slide 1

Begin Slide 2
  If AudioCapable
     // Play audio clip in file interference.au
     Audio (interference.au)
  Else
     // Present the substitute text
     Text (Interference.txt); at (0, 300); 400 X 100

  // Pause for 5 seconds before proceeding
  Wait(5)
  // Present a text part, an image containing one white and one blue pointer,
  //  and an image containing one yellow and one red pointer all in parallel.
  Begin Parallel
      Text (Solution1.txt); at (300, 300); 200 X 100
      // present Solution1.gif image at regular size for the image:
      Image(Solution1.gif) at (0,0)
      Pointer(white) at (178, 102)
      Pointer(blue) at 270, 71)
      Image(Solution2.gif) at (250, 0); 150 X 150
      Pointer(yellow) at (302, 76)
      Pointer(red) at (357, 36)
  End Parallel
End Slide 2

Begin Slide 3
 // Present a text part and an image containing a green pointer all in parallel
```

```
    Begin Parallel
        Text(Closeup.txt) at (50, 300); 300 X 100
        Image(Solution3.gif) at (0,0)
        Pointer(green) at (134, 117)
    End Parallel
    Wait(7)
    Image(Closeup.gif) at (300, 250); 200 X 150
End Slide 3

// The following three statements belong to a non-slide part:
Image(P24.gif) at (0,0)
Image(P25.gif) at (0, 300)
Wait(8)

Begin Slide 4
    Audio(Closing.au)
    Wait(2)
    // Present a text part and an image containing no pointers in parallel
    Begin Parallel
        Text(Closing.txt) at (0, 0) 400 X 150
        Image(ThankYou.gif) at (0, 150)
    End Parallel
End Slide 4
```

Scenario Code

The applet that implements this example scenario is given below. The code shown in bold faced print represents code generated by the Scenario Mail program. The regular print represents template code.

```
import java.awt.*;
import java.applet.*;
import java.awt.image.*;
import java.net.*;
import java.io.*;
import java.net.MalformedURLException;

public class ExampleScen extends Applet implements Runnable
{
    private boolean RetVal;                    // Value returned by method call
    private Button Previous;                   // Button used to move to previous slide
    private Button Next;                       // Button used to move to next slide
```

```
private MediaTracker tracker;                  // Tracks the status of image objects
private boolean AudioCapable;                  // False if no sound capability is available
public Thread ScenarioThread = null;           // applet thread
private int state = 0;                         // Indicates the part of the message
                                               // currently being played
private int ImageState = 0;                    // Indicates part of state consisting of
                                               // the group of images prior to the next
                                               // wait
private TextArea TextBox;                      // Used to display a text part of a message
static final int SCENARIO_DONE = -999;         // End of message sentinel
static final int ButtonWidth = 70;             // Width of next and previous buttons
static final int ButtonHeight = 35;            // Height of next and previous buttons

//* Scenario specific declarations will follow
private Image image0;
private AudioClip audio0;
private Image image1;
private Image image2;
private Image image3;
private Image image4;
private Image image5;
private Image image6;
private AudioClip audio1;
private Image image7;
static final int NumStates = 5;
static final int PScale = 15;
static final int PrevBtnXPos = 560;
static final int PrevBtnYPos = 450;
static final int NextBtnXPos = 640;

public void init()
{
  URL TextData = null;                         // Description of textfile Audio.cntl
  InputStream AudioCntlStream = null;          // Audio.cntl stream to get audio flag from
  char AudioFlag = 'Y';                        // "N" for not audio capable

  // Start at first part of message being played
  state = 0;
  tracker = new MediaTracker(this);

  // No flow layout manager for applet frame
  setLayout(null);

  // Add next and previous buttons to the frame - Each will be visible
  // as appropriate
  Previous = new Button("Previous");
```

```
    add(Previous);
    Previous.reshape(PrevBtnXPos, PrevBtnYPos, ButtonWidth, ButtonHeight);
    Next = new Button("Next");
    add(Next);
    Next.reshape(NextBtnXPos, PrevBtnYPos, ButtonWidth, ButtonHeight);
    Previous.hide();
    Next.hide();

// determine if the presenting machine has sound capability
    AudioCapable = true;    // assume sound capable until proven false

    // prepare the text file for reading:
    try
    {
       TextData = new URL(getCodeBase(), "Audio.cntl");
    }
    catch (Exception e)
    {
       showStatus("Audio.cntl file error ==> Assuming Audio Capable");
    }

    if (TextData != null)
    {
       try
       {
          AudioCntlStream = TextData.openStream();
       }
       catch(IOException e)
       {
          showStatus("Audio.cntl file error ==> Assuming Audio Capable");
       }

       if (AudioCntlStream != null)
       {
          // get the flag value from the file
          try
          {
             AudioFlag = (char) AudioCntlStream.read();
          }
          catch(IOException e)
          {
             showStatus("Audio.cntl file error ==> Assuming Audio Capable");
          }

          // if file contains flag value of "N", then machine is not audio capable
```

```
        if (AudioFlag != -1)
        {
          if ((AudioFlag == 'n') || (AudioFlag == 'N'))
            AudioCapable = false;
        }
      }
    }
}

public boolean action(Event evt, Object arg)
{
  // if next button was clicked move to the next slide

 if (arg.equals("Done"))
   System.exit(0);

 else if (arg.equals("Next"))
 {
    if (state == NumStates)
      state = SCENARIO_DONE;
    else
      state += 1;

    if (state > 1)
       Previous.hide();

    Next.hide();

    // Continue with rest of scenario:
    ScenarioThread.resume();
 }

 // if next button was clicked move to the previous slide
 else if (arg.equals("Previous"))
 {
    if (state > 0)
      state -= 1;

    // Continue with rest of scenario:
    ScenarioThread.resume();
  }

  return true;
}
```

```
public void run()
{
  // Prepare text window used to display all text parts of message:
  TextBox = new TextArea(8, 40);
  add("Center", TextBox);

  // While the entire scenario has not been played, play the next part of the
  // message (scenario):
  while (state < NumStates)
  {
     if (state < (NumStates - 1))
        Next.setLabel("Next");
     else
        Next.setLabel("Done");

     if (state == 0)
     {
        ImageState = 0;
        repaint();              // Show images and pointers for this part
        ShowText(TextBox, "CircProb.txt", 0, 300, 400, 100);
        try {ScenarioThread.sleep(3000);}
        catch (InterruptedException e) {}
        ImageState++;
        repaint();
        Next.show();
        ScenarioThread.suspend();
        TextBox.hide();
       Next.hide();
      }
     else if(state == 1)
     {
        getAppletContext().showStatus(" ");
        ImageState = 0;
        repaint();
        if (AudioCapable != false)
        {
           audio0 = getAudioClip(getCodeBase(), "interference.au");
           audio0.play();
           getAppletContext().showStatus("Audio: interference.au; ");
        }
        if (AudioCapable == false)
           ShowText(TextBox, "Interference.txt", 0, 300, 400, 100);
        try {ScenarioThread.sleep(5000);}
        catch (InterruptedException e) {}
        ShowText(TextBox, "Solution1.txt", 300, 300, 200, 100);
        ImageState++;
```

```
    repaint();
    Previous.show();
    Next.show();
    ScenarioThread.suspend();
    TextBox.hide();
    Previous.hide();
    Next.hide();
  }
  else if(state == 2)
  {
    getAppletContext().showStatus(" ");
    ImageState = 0;
    repaint();
    ShowText(TextBox, "Closeup.txt", 50, 300, 300, 100);
    try {ScenarioThread.sleep(7000);}
    catch (InterruptedException e) {}
    ImageState++;
    repaint();
    Previous.show();
    Next.show();
    ScenarioThread.suspend();
    TextBox.hide();
    Previous.hide();
   Next.hide();
  }
  else if(state == 3)
  {
    getAppletContext().showStatus(" ");
    ImageState = 0;
    repaint();
    try {ScenarioThread.sleep(8000);}
    catch (InterruptedException e) {}
    TextBox.hide();
    state = state + 1;
   }
  else if(state == 4)
  {
    getAppletContext().showStatus(" ");
    ImageState = 0;
    repaint();
    if (AudioCapable != false)
    {
      audio1 = getAudioClip(getCodeBase(), "Closing.au");
      audio1.play();
      getAppletContext().showStatus("Audio: Closing.au; ");
    }
```

```
            try {ScenarioThread.sleep(2000);}
            catch (InterruptedException e) {}
            ShowText(TextBox, "Closing.txt", 0, 0, 400, 150);
            ImageState++;
            repaint();
            Previous.show();
            Next.show();
            ScenarioThread.suspend();
            TextBox.hide();
            Previous.hide();
            Next.hide();
        }
    }
}

public void paint(Graphics g)
{
    // The code for displaying the images and pointers for each state (phase)
    // of the scenario will follow:
    if (state == 0)
    {
        if (ImageState == 0)
        {
        }
        else if (ImageState == 1)
        {
            image0 = getImage(getCodeBase(), "CircProb.jpeg");
            tracker.addImage(image0, 0);
            try
            {
                tracker.waitForID(0);
            }
            catch(InterruptedException e)
            {
            }
            RetVal = g.drawImage(image0, 0, 0, 200, 200, this);
            ShowPointer(g, 174, 76, Color.white);
            ShowPointer(g, 24, 38, Color.black);
        }
    }
    else if(state == 1)
    {
        if (ImageState == 0)
        {
        }
        else if (ImageState == 1)
```

```
    {
      image1 = getImage(getCodeBase(), "Solution1.gif");
      tracker.addImage(image1, 1);
      try
      {
         tracker.waitForID(1);
      }
      catch(InterruptedException e)
      {
      }
      RetVal = g.drawImage(image1, 0, 0, this);
      ShowPointer(g, 178, 102, Color.white);
      ShowPointer(g, 270, 71, Color.blue);
      image2 = getImage(getCodeBase(), "Solution2.gif");
      tracker.addImage(image2, 2);
      try
      {
         tracker.waitForID(2);
      }
      catch(InterruptedException e)
      {
      }
      RetVal = g.drawImage(image2, 250, 0, 150, 150, this);
      ShowPointer(g, 302, 76, Color.yellow);
      ShowPointer(g, 357, 36, Color.red);
    }
  }
  else if(state == 2)
  {
    if (ImageState == 0)
    {
      image3 = getImage(getCodeBase(), "Solution3.gif");
      tracker.addImage(image3, 3);
      try
      {
         tracker.waitForID(3);
      }
      catch(InterruptedException e)
      {
      }
      RetVal = g.drawImage(image3, 0, 0, this);
      ShowPointer(g, 134, 117, Color.green);
    }
    else if (ImageState == 1)
    {
      image4 = getImage(getCodeBase(), "Closeup.gif");
```

```
        tracker.addImage(image4, 4);
        try
        {
            tracker.waitForID(4);
        }
        catch(InterruptedException e)
        {
        }
        RetVal = g.drawImage(image4, 300, 250, 200, 150, this);
    }
}
else if(state == 3)
{
    if (ImageState == 0)
    {
        image5 = getImage(getCodeBase(), "P24.gif");
        tracker.addImage(image5, 5);
        try
        {
            tracker.waitForID(5);
        }
        catch(InterruptedException e)
        {
        }
        RetVal = g.drawImage(image5, 0, 0, this);
        image6 = getImage(getCodeBase(), "P25.gif");
        tracker.addImage(image6, 6);
        try
        {
            tracker.waitForID(6);
        }
        catch(InterruptedException e)
        {
        }
        RetVal = g.drawImage(image6, 0, 300, this);
    }
}
else if(state == 4)
{
    if (ImageState == 0)
    {
    }
    else if (ImageState == 1)
    {
        image7 = getImage(getCodeBase(), "ThankYou.gif");
        tracker.addImage(image7, 7);
```

```
            try
            {
               tracker.waitForID(7);
            }
            catch(InterruptedException e)
            {
            }
            RetVal = g.drawImage(image7, 0, 150, this);
         }
      }
   }

   public void start()
    {
      //* Create the main thread for the applet
      if (ScenarioThread == null)
      {
         ScenarioThread = new Thread(this);
         ScenarioThread.start();
      }
   }

   void ShowText(TextArea TextBox, String TextFile, int x, int y, int width, int height)
   //*************************************************************************
   // Display the contents of a text file whose name is given by
   // parameter TextFile in a text window specified by parameter TextBox.
   // The location to display the text window is given by parameters x
   // and y.  The window will have height and width specified by
   // parameters width and height.
   //
   // Parameters:
   //      TextBox: Window to display text in
   //      TextFile: File containing text to display
   //      x: x location of upper left corner of TextBox.
   //      y: y location of upper left corner of TextBox.
   //      width: width of TextBox
   //      height: width of TextBox
   //*************************************************************************
   {
      boolean EndFile;                 // true when at end of textfile
      int ch=' ';                      // current char in textfile
      String Str;                      // string to add to textbox
      URL TextData = null;             // Description of textfile
      InputStream TextStream = null;   // stream to get text from
```

```
//* display text window:
TextBox.reshape(x, y, width, height);
TextBox.show();


//* prepare the text file for reading:
try
{
   TextData = new URL(getCodeBase(), TextFile);
}
catch (MalformedURLException e)
{
   System.out.println("Error " + e);
}

try
{
   TextStream = TextData.openStream();
}
catch(IOException e)
{
   showStatus("Cannot Display Contents of Text File: " + TextFile);
}


//* erase any text from a previous part of scenario
TextBox.selectAll();
TextBox.replaceText("", 0, TextBox.getSelectionEnd());

EndFile = false;

//* while not at end of TextFile, add the current character from the
//* file to the text window and move to next char in file
while (EndFile == false)
{
   try
   {
     ch = TextStream.read();
   }
   catch(IOException e)
   {
      showStatus("Error " + e);
   }

  if (ch != -1)
```

```
        {
          Str = "" + (char)ch;
          TextBox.appendText(Str);
        }
        else
          EndFile = true;
      }
    }

    void ShowPointer(Graphics g, int x, int y, Color PColor)
    //*****************************************************************
    // Display a pointer of color PColor at location x, y.
    //
    // Parameters:
    //      x: x location to display pointer
    //      y: y location to display pointer
    //      Pcolor: color to color interior of pointer
    //*****************************************************************
    {
      Polygon Pointer = new Polygon();   // polygon used for pointer
      int xPos, yPos;                    // coordinates of next point
                                         // to plot when displaying pointer

// Pointer will be composed of 7 points.  These 7 points are labeled 1 to 7
// as each point is added in counterclockwise order starting with the point
// at the "point" of the pointer.

//1 - Add point at "point" of pointer
      Pointer.addPoint(x, y);

//2 - Add next point
      xPos = x - (int)(0.07 * PScale);
      yPos = y + (int)(1.1 * PScale);
      Pointer.addPoint(xPos, yPos);

//3 - Add next point
      xPos = x - (int)(0.2 * PScale);
      yPos = y + (int)(0.8 * PScale);
      Pointer.addPoint(xPos, yPos);

//4 - Add next point
      xPos = x - (int)(0.6 * PScale);
      yPos = y + (int)(1.5 * PScale);
      Pointer.addPoint(xPos,  yPos);
```

```
//5 - Add next point
        xPos = x - (int)(0.9 * PScale);
        yPos = y + (int)(1.3 * PScale);
        Pointer.addPoint(xPos, yPos);

//6 - Add next point
        xPos = x - (int)(PScale/2);
        yPos = y + (int)(0.7 * PScale);
        Pointer.addPoint(xPos, yPos);

//7 - Add next point
        xPos = x - (int)(0.8 * PScale);
        yPos = y + (int)(0.8 * PScale);
        Pointer.addPoint(xPos, yPos);

        // draw black outline around pointer:
        g.setColor(Color.black);
        g.drawPolygon(Pointer);

        // fill interior with color PColor
        g.setColor(PColor);
        g.fillPolygon(Pointer);
    }
}
```

VITA

Jeffrey B. Holland

Candidate for the Degree of

Master of Science

Thesis: INTERACTIVE MULTIMEDIA MAIL SCENARIOS: AN IMPLEMENTATION

Major Field: Computer Science

Biographical:

Personal Data: Born in Muskogee, OK, On June 5, 1970, the son of Linda and Leon Holland.

Education: Graduated from Coweta High School, Coweta, Oklahoma in May 1988; received Associate in Arts in Liberal Arts degree from Tulsa Junior College, Tulsa, Oklahoma in July 1992; received a Bachelor of Science degree in Computer Science from Oklahoma State University, Stillwater, Oklahoma in December 1994. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University, Stillwater Oklahoma in July 1997.

Experience: Employed by Phillips Petroleum Company in Bartlesville, Oklahoma as a systems analyst/programmer from December 1996 to present. Employed by Oklahoma State University, Department of Computer Science in Stillwater, Oklahoma as a graduate teaching assistant from January 1995 to December 1996. Employed by Mercruiser Corporation in Stillwater, Oklahoma as a contract database programmer from September 1995 to March 1996. Employed by TMS, Inc. in Stillwater, Oklahoma as a software tester from December 1994 to January 1995 and as a software engineering apprentice from May 1995 to August 1995.